Application Note

Document No.: AN1138

G32R501 Migration Guide

Version: V1.1

1. Introduction

For embedded system designers, it is crucial to easily migrate applications between different series of MCU. With continuous increase in product demand, the demand for resources such as memory size and I/O quantity also increases, so designers often need to port applications to the MCU with higher performance or more resources.

This Migration Guide is intended to help users to analyze the steps required to migrate from existing Txx320F28004x device to G32R501 device. This document collects the most important information and lists the key matters needing attention in the migration process. The main aspects involved in the migration process include hardware porting, peripheral porting, firmware porting, and tool chain porting.

To make full use of the information in this guide, users shall be familiar with the characteristics and development environment of G32R501 series MCU. Users can refer to the following technical documents:

- G32R501 Series User Manual, Datasheet and Programming Manual
- Related documents for G32R501 series core, including Arm v8.1-M Architecture Reference Manual, etc.

This document will systematically introduce the steps of migration from Txx320F28004x to G32R501, and provide practical examples and codes to help designers to smoothly complete the application migration work and improve the overall performance and reliability of the system. The MCU applicable to this application note is G32R501.

Through this Migration Guide, the designers can better understand and address various challenges encountered in the migration process, to ensure that optimal performance of applications can be implemented on the new MCU platform.

1.1. Full Name and Abbreviation Description of Terms

The keywords and descriptions used in this document are as shown in Table 1.

Table 1 Abbreviation Description

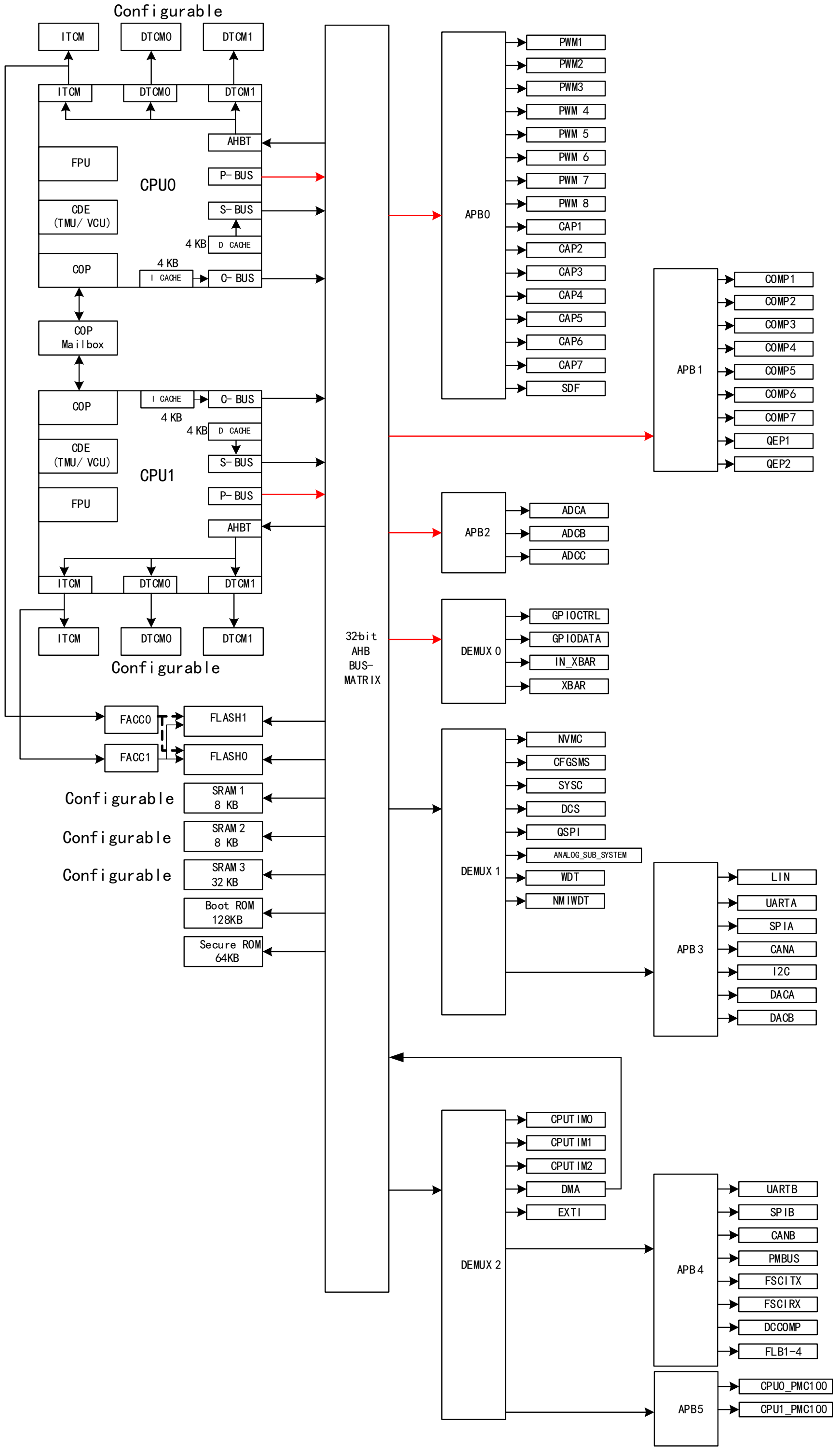
Full name in Chinese	Full name in English	English abbreviation
Configurable Static Memory Subsystem	Configurable Static Memory Subsystem	CFGSMS
Software	Software	SW
Hardware	Hardware	HW
Interface	Interface	IF
AHB slave interface	AHB slave interface	ahbs_if

1.2. R5xx SoC Introduction

R5xx SoC is a system on a chip (SoC) based on Arm® Cortex®-M52 dual cores. This SoC adopts the widely used AMBA 2.0 bus to integrate peripheral IP components: the components that require broad bandwidth are connected to the AMBA AHB bus, while the low-speed components are connected to the AMBA APB bus. DMA cannot access the registers related to FLB.

The architecture of R5xx SoC is as shown in Figure 1.

Figure 1 R5xx SoC Architecture



Note:

The red lines in the figure represent bus bridge.

The size of ITCM/DTCM/SRAM is configured using CFGSMS.

Contents

2. Boot Mode Compatibility

2.1. Boot Mode

This section introduces the Boot modes supported by G32R501. The G32R501 Boot mode is consistent with the competitor’s Txx320F28004x, and adds the Secure Boot function, which is consistent with the competitor’s Txx320F2838x.

BootROM uses the GPIO pins for boot control to determine the boot mode configuration. The device can be configured to boot to RAM, boot to Flash, execute bootloader, or remain in wait mode.

The following table shows the default boot mode options, and users can customize the supported boot modes and boot mode selection pins.

Table 2 Default Boot Modes of G32R501

Boot Mode	**GPIO24** **(Default boot mode select pin 1)**	**GPIO32** **(Default boot mode select pin 0)**
Parallel IO	0	0
UART / Wait Boot	0	1
CAN	1	0
Flash	1	1

The following table shows all Boot modes supported by G32R501. Users can customize the required boot mode and boot mode selection pin according to the project needs.

Table 3 All Available Boot Modes of G32R501

Boot Mode Number	**Boot Mode**
0	Parallel IO
1	UART / Wait Boot
2	CAN
3	Flash
4	Wait
5	RAM
6	SPI Master
7	I2C Master
10	Secure Boot

2.2. GPIO Pin Allocation in Boot Mode

Users can choose different GPIO pins according to the package and other configuration information.

Table 4 UART Boot

Option	BootMode Value	Tx	Rx
0	0x01	GPIO29	GPIO28
1	0x21	GPIO16	GPIO17
2	0x41	GPIO8	GPIO9
3	0x61	GPIO48	GPIO49
4	0x81	GPIO24	GPIO25

Table 5 CAN Boot

Option	BootMode Value	Tx	Rx
0	0x02/82	GPIO32	GPIO33
1	0x22/A2	GPIO4	GPIO5
2	0x42/C2	GPIO31	GPIO30
3	0x62/E2	GPIO37	GPIO35

Table 6 Flash Boot

Option	BootMode Value	Flash Entry Point	Flash Bank, Sector
0	0x03	0x08000000	Bank 0, Sector 0 (Busmatrix IF)
1	0x23	0x00100000	Bank 0, Sector 0 (ITCM IF)
2	0x43	0x08200000	Bank 1, Sector 0 (Busmatrix IF, DualBank Mode)
3	0x63	0x00120000	Bank 1, Sector 0 (ITCM IF, BualBank Mode)

Table 7 Wait Boot

Option	BootMode Value	Watchdog Status
0	0x04	Enable
1	0x24	Disable

Table 8 RAM Boot

Option	BootMode Value	RAM Entry Point	RAM IF
0	0x05	0x00000000	ITCM
1	0x25	0x20100000	SRAM0
2	0x45	0x20200000	SRAM1
3	0x65	0x20300000	SRAM2

Table 9 SPI Boot

Option	BootMode Value	MOSI	MISO	SCK	CS
1	0x26	GPIO8	GPIO10	GPIO9	GPIO10
2	0x46	GPIO54	GPIO55	GPIO56	GPIO57
3	0x66	GPIO16	GPIO17	GPIO56	GPIO57
4	0x86	GPIO8	GPIO17	GPIO9	GPIO11

Table 10 I2C Boot

Option	BootMode Value	SDA	SCL
0	0x07	GPIO32	GPIO33
2	0x47	GPIO26	GPIO27
3	0x67	GPIO42	GPIO43

Table 11 Secure Boot

Option	BootMode Value	Flash Bank, Sector
0	0x0A	Bank 0, Sector 0 (Busmatrix IF)
1	0x2A	Bank 0, Sector 0 (ITCM IF)
2	0x4A	Bank 1, Sector 0 (Busmatrix IF, DualBank Mode)
3	0x6A	Bank 1, Sector 0 (ITCM IF, BualBank Mode)

2.2.1. Example 1: Using GPIO3/4/5 to select different boot modes

This example demonstrates how to configure GPIO3, GPIO4 and GPIO5 as the boot mode selection pin to achieve eight (e.g. Table 12) different system boot modes. Through combination of different pin levels, users can flexibly choose flash, UART, and CAN boot modes.

Table 12 System Boot Mode 1

Table	BMSP0	BMSP1	BMSP2	BootMode
0	0	0	0	Flash boot (0x03)
1	1	0	0	UART boot (0x01)
2	0	1	0	Flash boot (0x23)
3	1	1	0	UART boot (0x21)
4	0	0	1	CAN boot (0x02)
5	1	0	1	SPI boot (0x06)
6	0	1	1	RAM boot (0x05)
7	1	1	1	I2C boot (0x07)

In the following codes:

- The address of EMU_BOOTPIN_CONFIG is 0x50020000
- The address of EMU BOOTDEF LOW is 0x50020004
- The address of EMU BOOTDEF HIGH is 0x50020008

Configuration steps:

Configure GPIO3/4/5 as BootModePin:

```
HWREG(EMU_BOOTPIN_CONFIG) = (BOOTPIN_CONFIG_KEY    << 24) +
                            (BOOTPIN_CONFIG_BMSP2 << 16) +
                            (BOOTPIN_CONFIG_BMSP1 << 8) +
                            (BOOTPIN_CONFIG_BMSP0);
```

- Write 0x5A to the Key field of EMU_BOOTPIN_CONFIG to perform a custom boot process.
- Set the BMSP value:
  - BOOTPIN_CONFIG_BMSP0 = 0x03
  - BOOTPIN_CONFIG_BMSP1 = 0x04
  - BOOTPIN_CONFIG_BMSP2 = 0x05

(1) Write the boot mode options 0, 1, 2, and 3 into the EMU_BOOTDEF_LOW register:

```
HWREG(EMU_BOOTDEF_LOW) = (BOOTDEF_LOW_3 << 24) +
                         (BOOTDEF_LOW_2 << 16) +
                         (BOOTDEF_LOW_1 << 8) +
                         (BOOTDEF_LOW_0);
```

- BOOTDEF_LOW_0 = 0x03 (flash boot)
- BOOTDEF_LOW_1 = 0x01 (UART boot)
- BOOTDEF_LOW_2 = 0x23 (flash boot)
- BOOTDEF_LOW_3 = 0x21 (UART boot)

(2) Write the boot mode options 4, 5, 6, and 7 into the EMU_BOOTDEF_HIGH register:

```
HWREG(EMU_BOOTDEF_HIGH) = (BOOTDEF_HIGH_7 << 24) +
                          (BOOTDEF_HIGH_6 << 16) +
                          (BOOTDEF_HIGH_5 << 8) +
                          (BOOTDEF_HIGH_4);
```

- BOOTDEF_HIGH_4 = 0x02 (CAN boot)
- BOOTDEF_HIGH_5 = 0x06 (SPI boot)

- BOOTDEF_HIGH_6 = 0x05 (RAM boot)
- BOOTDEF_HIGH_7 = 0x07 (I2C boot)

After completing the above configuration, different boot modes can be selected by controlling the external level of GPIO3/4/5 in simulation state.

2.2.2. Example 2: Using GPIO3/4 to select different boot modes

In this example, by configuring GPIO3 and GPIO4 as the boot mode selection pins, four (e.g. Table 13) different system boot modes can be achieved. This method is applicable to the situations where boot mode selection needs to be simplified and few pins are used.

Table 13 System Boot Mode 2

Table	BMSP1	BMSP0	BootMode
0	0	0	Flash boot (0x03)
1	0	1	UART boot (0x01)
2	1	0	RAM boot (0x05)
3	1	1	CAN boot (0x02)

Based on the configuration steps in Example 1,

(1) Configure GPIO3/4 as BootModePin:

- Write 0x5A to the Key field of EMU_BOOTPIN_CONFIG to perform a custom boot process.
- Set the BMSP value:
  - BOOTPIN_CONFIG_BMSP0 = 0x03
  - BOOTPIN_CONFIG_BMSP1 = 0x04
  - BOOTPIN_CONFIG_BMSP2 = 0xFF (not used)

(2) Write the boot mode options 0, 1, 2, and 3 into the EMU_BOOTDEF_LOW register:

- BOOTDEF_LOW_0 = 0x03 (flash boot)
- BOOTDEF_LOW_1 = 0x01 (UART boot)
- BOOTDEF_LOW_2 = 0x05 (RAM boot)
- BOOTDEF_LOW_3 = 0x02 (CAN boot)

(3) Write the effective boot mode option into the EMU_BOOTDEF_HIGH register:

- BOOTDEF_HIGH_4 = 0xFF
- BOOTDEF_HIGH_5 = 0xFF

- BOOTDEF_HIGH_6 = 0xFF
- BOOTDEF_HIGH_7 = 0xFF

After completing the above configuration, different boot modes can be selected by controlling the external level of GPIO3/4 in simulation state.

2.3. BootMode State

BootMode is the running result of Boot mode configuration.

Base address: 0x2030_7F00

Structure:

Table 14 Current Boot Mode

Bit	Name	Desc
[31:8]	RESERVE	Reserved
[7:5]	Boot Option	Used to indicate the configuration of current boot mode
[4:3]	RESERVE	Reserved
[2:0]	Boot Mode	Used to indicate the current boot mode: 0: Parallel IO 1: UART / Wait boot 2: CAN 3: Flash 4: Wait 5: RAM 6: SPI Master 7: I2C Master 10: Secure Boot

2.4. Boot Execution Progress Information

Users can confirm the progress information during Boot execution according to the setting information of this register.

Base address: 0x2030_7F08

Structure:

Table 15 Boot Execution Progress Information

Bit	Status	Desc
31	BOOTROM_IN_HARDFULT_HANDLER	Boot enters the Hard Fault handler
[30:28]	/	RESERVE

Bit	Status	Desc
27	BOOTROM_GOT_A_HARD_FAULT	Boot generates a Hard Fault
26	BOOTROM_GOT_A_USAGE_FAULT	Boot generates a Usage Fault
25	BOOTROM_GOT_A_BUS_FAULT	Boot generates a Bus Fault
24	BOOTROM_GOT_A_MEM_FAULT	Boot generates a Memory Management Fault
23	/	RESERVE
22	BOOTROM_GOT_A_MCLK_NMI	Boot generates a clock loss NMI
21	/	RESERVE
20	BOOTROM_GOT_A_FLASH_UNCERR_NMI	Boot generated a Flash ECC uncorrectable error NMI
[19:16]	/	RESERVE
15	BOOTROM_BOOT_COMPLETE	Boot is completed
14	/	RESERVE
13	BOOTROM_HANDLED_POR	Boot is clearing POR flag bit
12	BOOTROM_HANDLED_XRSN	Boot is clearing XRSN flag bit
11	BOOTROM_RESC_HANDLED	Boot is handling the reset flag
10	BOOTROM_POR_MEM_TEST_DONE	Boot has completed MBIST
9	/	RESERVE
8	BOOTROM_IN_SECURE_BOOT	Boot enters Secure BOOT
7	BOOTROM_IN_CAN_BOOT	Boot enters CAN BOOT
6	BOOTROM_IN_I2C_BOOT	Boot enters I2C BOOT
5	BOOTROM_IN_SPI_BOOT	Boot enters SPI BOOT
4	BOOTROM_IN_UART_BOOT	Boot enters SCI BOOT
3	BOOTROM_IN_RAM_BOOT	Boot enters RAM BOOT
2	BOOTROM_IN_PARALLEL_BOOT	Boot enters PARALLEL IO BOOT
1	BOOTROM_IN_FLASH_BOOT	Boot enters FLASH BOOT
0	BOOTROM_START_BOOT	Start the boot process

2.5. Boot Execution Exception Information

Table 16 Boot Execution Exception Information

Addr	Name	Desc
0x20307F0C	hard_fault_status	Record the SCB_HFSR state when entering Hard Fault
0x20307F10	cbrom_fault_status	Record the NMI_FLG when entering NMI
0x20307F14	bus_fault_address	Record the SCB_BFAR state when entering Hard Fault
0x20307F18	bus_fault_status	Record the SCB_BFSR state when entering Hard Fault

Addr	Name	Desc
0x20307F1C	mem_manage_fault_address	Record the SCB_MMFAR state when entering Hard Fault
0x20307F20	mem_manage_fault_status	Record the SCB_MMFSR state when entering Hard Fault
0x20307F24	usage_fault_status	Record the SCB_UFSR state when entering Hard Fault

2.6. Boot Wait Point

During ROM execution, the CPU may enter a waiting loop state in the code. This state may occur for various reasons. The address range in which the register value of CPU program counter (PC) falls is listed in Table 17, and users can judge whether Boot has entered an exception by reading the current PC address.

Table 17 Address Information of Boot Waiting Point

Entry	G32R501 Wait point	Txx320F28004x Wait point
In Wait Boot	0x1000_17DC—0x1000_17F4	0x003FAD74—0x003FB0CD
In UART Boot	0x1001_1DC4—0x1001_1E14	0x000706DC—0x000706DF
In Hard Fault Handler	0x1000_0212—0x1000_0312	—
In NMI Handler	0x1000_0316—0x1000_044E	0x003FBCD1—0x003FBD67
Failed secure Flash CMAC verification loop	0x1001_0F80—0x10010FB8	—

3. Peripheral Transplantation

It is crucial to understand the differences in peripheral resources between the two MCU in the migration process from Txx320F28004x to G32R501. The selection and configuration of peripheral resources not only directly affects the performance and function of the system, but also concerns the implementation and optimization of applications. This chapter will compare the peripheral resources of the two MCU in detail.

3.1. Resource Comparison List

To clearly display the differences in peripheral resources between Txx320F28004x and G32R501, the following list compares the main peripheral resources of these two microcontrollers.

Table 18 Peripheral Resource Differences between G32R501 and Txx320F28004x

Characteristics		**G32R501**	**Txx320F28004x**
Processor and accelerator			
CPU0	Core	Cortex-M52	C28x
	Frequency	200MHz (xxx7 device model)	100MHz
	FPU	Supported	Supported
	TMU	Supported (mathematical calculation acceleration instruction set)	Supported
	VCU	Supported (mathematical calculation acceleration instruction set)	Supported
CPU1 (xxxD device model)	Core	Cortex-M52	CLA
	Frequency	200MHz (xxx7 device model)	100MHz
	FPU	Supported	The core is floating point type, without independent FPU
	TMU	Supported (mathematical calculation acceleration instruction set)	Not supported
	VCU	Supported (mathematical calculation acceleration instruction set)	Not supported
	6-channel DMA	Supported	Supported
Memory			
Flash		640KB	256KB
RAM		128KB	100 KB
Code Storage Area Security Protection (DCSM)		Supported	Supported
RAM Parity		Supported	Supported (except M0, M1)
RAM ECC		Not supported	Supported (M0, M1 only)

Characteristics		G32R501	Txx320F28004x
FLASH ECC		Supported	Supported
Bootstrap ROM		Supported	Supported
User-configurable DCSM OTP		8KB	4KB
System			
Logic block		4 logical blocks (FLB)	4 logical blocks (CLB)
32-bit CPU timer		3	3
Watchdog timer		1	1
Non-maskable interrupt watchdog (NMIWD) timer		1	1
Crystal oscillator/External clock input		1	1
Internal pin oscillator		2	2
GPIO pin	LQFP100	42	40
	LQFP80	44	/
	LQFP64	26	26
	VQFN56	25	25
AIO input	LQFP100	31	21
	LQFP80	16	/
	LQFP64	16	14
	VQFN56	14	12
	External Interrupt	16	5
Analog peripherals			
ADC	Accuracy	12 bits	12 bits
	Quantity of ADC	3	3
	Million Samples Per Second (MSPS)	3.45	3.45
	Conversion time (ns)	290	290
ADC channel (single-end)	LQFP100	31	21
	LQFP80	16	/
	LQFP64	16	14
	QFN56	14	12
Temperature sensor		1	1
Buffer DAC		2	2

Characteristics		G32R501	Txx320F28004x
COMP (enhanced comparator)	LQFP100	7	7
	LQFP80	4	/
	LQFP64	6	6
	QFN56	5	5
PGA		Not supported	Supported
Control peripherals			
CAP/HRCAP module		7 (2 with HRCAP function)	7 (2 with HRCAP function)
PWM/HRPWM channel		16	16
QEP module	LQFP100	2	2
	LQFP80	2	/
	LQFP64	2	1
	QFN56	2	1
SDF channel	LQFP100	4	4
	LQFP80	4	/
	LQFP64	2	2
	QFN56	2	2
Communication peripherals			
CAN		2	2
I2C		1	1
UART (UART-compatible)		2	2
SPI		2	2
LIN (UART-compatible)		1	1
PMBus		1	1
QSPI		1	0
Operating voltage and temperature			
Operating voltage range		3.1V~3.6V	3.1V~3.6V
Ambient operating temperature (TA)		From -40°C to 105℃ (xxx7 device model) From -40°C to 125°C (xxx8 device model)	From -40°C to 105℃ (xxxS device model) From -40°C to 125°C (xxxQ device model)

3.2. CPU Core

Compared with traditional microcontrollers, G32R501 is based on Arm® Cortex®-M52 core and additionally supports high-performance DSP instruction set. Arm® Cortex®-M52 integrates Arm v8.1-M instruction set, including fixed-point and floating-point DSP extension, enabling it to efficiently execute complex signal processing and control algorithms.

Note: During program development, the compiler is responsible for generating executable files for program codes developed using high-level language, while the code developed using assembly language needs to be manually modified according to the Arm v8.1-M instruction set.

3.2.1. Floating-point unit (FPU)

G32R501 supports single-precision and double-precision floating-point operations (optional), so it has a significant advantage in applications that require high-precision calculation. The FPU can significantly improve the speed and precision of floating-point arithmetic, and its performance of performing floating-point arithmetic is significantly improved compared with the cores without FPU.

3.2.2. Helium instruction set

G32R501 supports single-instruction multi-data (Helium) operations, which enables it to improve data processing efficiency in parallel processing. The Helium instruction set can process multiple data elements at a time, making it suitable for applications that require high data throughput such as image processing and audio processing.

3.2.3. Integer Computation Acceleration Unit (ICAU)

In the extension space CDE of G32R501, Zidian implements some specific instructions to improve the performance of integer arithmetic, including:

- Computation required by Helium: include Fast Fourier Transform (FFT), complex operation, etc.
- CRC algorithm: Improve the efficiency of cyclic redundancy check (CRC) calculation.

3.2.4. Floating-point Computation Acceleration Unit (FCAU)

In the extension space FPCDE of G32R501, Zidian implements some specific instructions to improve the performance of floating-point arithmetic, including:

- Trigonometric function calculation: Improve the efficiency of trigonometric function operation.
- Square root calculation: Improve the speed of square root calculation.
- Division operation: Improve the performance of floating-point division.

3.3. Interrupt Vector Table Differences

The interrupt numbers used by G32R501 can support up to 226 interrupt lines. All interrupts are sent to the CPU through the nested vector interrupt controller (NVIC). The external interrupt controller (EXTI) provides a total of 16 external interrupts (GPIO/INPUT_XBAR) and peripheral interrupts (COMP).

Table 19 Comparison of Interrupts and Abnormal Functions

Function	G32R501	Txx320F28004x
Interrupt priority	226	19 (ePIE extended to 224)
Interrupt nesting	8-level hardware nesting	No (software support)
Number of interrupt lines	226	19 (ePIE extended to 224)
Interrupt response delay (cycle)	15	37

In the interrupt vector table, G32R501 and Txx320F28004x are not the same, and for the differences between them, refer to Table 20 Interrupt Vector Differences.

Table 20 Interrupt Vector Differences

Vector ID	G32R501 Name	Txx320F28004x
-15	Reset	Not used
-14	NMI	Not used
-13	HardFault	Not used
-12	MemManage	Not used
-11	BusFault	Not used
-10	UsageFault	Not used
-9	SecureFault	Not used
-5	SVCall	Not used
-4	Debug Monitor	Not used
-2	PendSV	Not used
-1	SysTick	Not used
0	Reserved	Not used
1	Reserved	Not used
2	Reserved	Not used
3	Reserved	Not used
4	Reserved	Not used
5	Reserved	Not used
6	Reserved	Not used
7	Reserved	Not used
8	Reserved	Not used
9	Reserved	Not used

Vector ID	G32R501 Name	Txx320F28004x
10	Reserved	Not used
11	DCCOMP interrupt	Not used
12	Reserved	Not used
13	TMR1	CPU TIMER1 Interrupt
14	TMR2	CPU TIMER2 Interrupt
15	Reserved	CPU Data Logging Interrupt
16	RAM	CPU Real-Time OS Interrupt
17	COP TE0	Reserved
18	COP TE1	Non-Maskable Interrupt
19	COP TE2	Illegal Instruction (ITRAP)
20	COP TE3	User-Defined Trap
21	COP RF0	User-Defined Trap
22	COP RF1	User-Defined Trap
23	COP RF2	User-Defined Trap
24	COP RF3	User-Defined Trap
25	COP GP0	User-Defined Trap
26	COP GP1	User-Defined Trap
27	COP GP2	User-Defined Trap
28	COP GP3	User-Defined Trap
29	Reserved	User-Defined Trap
30	Reserved	User-Defined Trap
31	Reserved	User-Defined Trap
32	ADCA1	ADCA1 interrupt
33	ADCB1	ADCB1 interrupt
34	ADCC1	ADCC1 interrupt
35	Reserved	XINT1 interrupt
36	Reserved	XINT2 interrupt
37	Reserved	Reserved
38	TMR0	TIMER0 interrupt
39	WWDT	WAKE WDOG interrupt
40	PWM1 TRIP ZONE	EPWM1 trip zone interrupt
41	PWM2 TRIP ZONE	EPWM2 trip zone interrupt

Vector ID	G32R501 Name	Txx320F28004x
42	PWM3 TRIP ZONE	EPWM3 trip zone interrupt
43	PWM4 TRIP ZONE	EPWM4 trip zone interrupt
44	PWM5 TRIP ZONE	EPWM5 trip zone interrupt
45	PWM6 TRIP ZONE	EPWM6 trip zone interrupt
46	PWM7 TRIP ZONE	EPWM7 trip zone interrupt
47	PWM8 TRIP ZONE	EPWM8 trip zone interrupt
48	PWM1	EPWM1 interrupt
49	PWM2	EPWM2 interrupt
50	PWM3	EPWM3 interrupt
51	PWM4	EPWM4 interrupt
52	PWM5	EPWM5 interrupt
53	PWM6	EPWM6 interrupt
54	PWM7	EPWM7 interrupt
55	PWM8	EPWM8 interrupt
56	CAP1	ECAP1 interrupt
57	CAP2	ECAP2 interrupt
58	CAP3	ECAP3 interrupt
59	CAP4	ECAP4 interrupt
60	CAP5	ECAP5 interrupt
61	CAP6	ECAP6 interrupt
62	CAP7	ECAP7 interrupt
63	Reserved	Reserved
64	QEP1	EQEP1 interrupt
65	QEP2	EQEP2 interrupt
66	Reserved	Reserved
67	Reserved	Reserved
68	FLB1	CLB1 interrupt
69	FLB2	CLB2 interrupt
70	FLB3	CLB3 interrupt
71	FLB4	CLB4 interrupt
72	SPIA_RX	SPIA_RX interrupt
73	SPIA_TX	SPIA_TX interrupt

Vector ID	G32R501 Name	Txx320F28004x
74	SPIB_RX	SPIB_RX interrupt
75	SPIB_TX	SPIB_TX interrupt
76	Reserved	Reserved
77	Reserved	Reserved
78	Reserved	Reserved
79	Reserved	Reserved
80	DMA_CH1	DMA_CH1 interrupt
81	DMA_CH2	DMA_CH2 interrupt
82	DMA_CH3	DMA_CH3 interrupt
83	DMA_CH4	DMA_CH4 interrupt
84	DMA_CH5	DMA_CH5 interrupt
85	DMA_CH6	DMA_CH6 interrupt
86	Reserved	Reserved
87	Reserved	Reserved
88	I2CA	I2CA interrupt
89	I2CA FIFO	I2CA FIFO interrupt
90	QSPI	Reserved
91	Reserved	Reserved
92	Reserved	Reserved
93	Reserved	Reserved
94	Reserved	Reserved
95	Reserved	Reserved
96	UARTA_RX	SCIA RX interrupt
97	UARTA_TX	SCIA TX interrupt
98	UARTB_RX	SCIB RX interrupt
99	UARTB_TX	SCIB TX interrupt
100	CANA INT0	CANA interrupt 0
101	CANA INT1	CANA interrupt 1
102	CANB INT0	CANB interrupt 0
103	CANA INT1	CANB interrupt 1
104	ADCA EVENT	ADCA event interrupt
105	ADCA2	ADCA2 interrupt

Vector ID	G32R501 Name	Txx320F28004x
106	ADCA3	ADCA3 interrupt
107	ADCA4	ADCA4 interrupt
108	ADCB EVENT	ADCB event interrupt
109	ADCB2	ADCB2 interrupt
110	ADCB3	ADCB3 interrupt
111	ADCB4	ADCB4 interrupt
112	EXTI_LINE0	CLA interrupt 1
113	EXTI_LINE1	CLA interrupt 2
114	EXTI_LINE2	CLA interrupt 3
115	EXTI_LINE3	CLA interrupt 4
116	EXTI_LINE4	CLA interrupt 5
117	EXTI_LINE5	CLA interrupt 6
118	EXTI_LINE6	CLA interrupt 7
119	EXTI_LINE7	CLA interrupt 8
120	EXTI_LINE8	XINT3 interrupt
121	EXTI_LINE9	XINT4 interrupt
122	EXTI_LINE10	XINT5 interrupt
123	EXTI_LINE11	Reserved
124	EXTI_LINE12	Reserved
125	EXTI_LINE13	Reserved
126	EXTI_LINE14	FPU overflow interrupt
127	EXTI_LINE15	FPU underflow interrupt
128	Reserved	Reserved
129	FPU DZC	Reserved
130	FPU IDC	Reserved
131	FPU IOC	Reserved
132	FPU OFC	Reserved
133	FPU UFC	Reserved
134	FPU IXC	Reserved
135	Reserved	Reserved
136	Reserved	Reserved
137	Reserved	Reserved

Vector ID	G32R501 Name	Txx320F28004x
138	Reserved	Reserved
139	Reserved	Reserved
140	Reserved	Reserved
141	Reserved	Reserved
142	Reserved	Reserved
143	Reserved	Reserved
144	Reserved	Reserved
145	Reserved	Reserved
146	Reserved	Reserved
147	Reserved	Reserved
148	Reserved	Reserved
149	Reserved	Reserved
150	Reserved	Reserved
151	Reserved	Reserved
152	Reserved	Reserved
153	Reserved	Reserved
154	Reserved	Reserved
155	Reserved	Reserved
156	Reserved	Reserved
157	CAP6 HRcalibration	ECAP6 HRcalibration interrupt
158	CAP7 HRcalibration	ECAP7 HRcalibration interrupt
159	Reserved	Reserved
160	SDF	SDFM1 interrupt
161	Reserved	Reserved
162	Reserved	Reserved
163	Reserved	Reserved
164	SDF DR INT1	SDFM1 DR interrupt 1
165	SDF DR INT2	SDFM1 DR interrupt 2
166	SDF DR INT3	SDFM1 DR interrupt 3
167	SDF DR INT4	SDFM1 Dr interrupt 4
168	Reserved	Reserved
169	Reserved	Reserved

Vector ID	G32R501 Name	Txx320F28004x
170	Reserved	Reserved
171	Reserved	Reserved
172	Reserved	Reserved
173	Reserved	Reserved
174	Reserved	Reserved
175	Reserved	Reserved
176	Reserved	Reserved
177	Reserved	Reserved
178	-	FSITX_INT1
179	-	FSITX_INT2
180	-	FSIRX_INT1
181	-	FSIRX_INT2
182	Reserved	CLAPROMCRC interrupt
183	Reserved	Reserved
184	LINA INT1	LINA interrupt 0
185	LINA INT2	LINA interrupt 1
186	Reserved	Reserved
187	Reserved	Reserved
188	PMBUSA interrupt	PMBUSA interrupt
189	Reserved	Reserved
190	Reserved	Reserved
191	Reserved	Reserved
192	Reserved	Reserved
193	Reserved	Reserved
194	Reserved	Reserved
195	Reserved	Reserved
196	Reserved	Reserved
197	Reserved	Reserved
198	Reserved	Reserved
199	Reserved	Reserved
200	ADCC EVENT	ADCC event interrupt
201	ADCC2	ADCC2 interrupt

Vector ID	G32R501 Name	Txx320F28004x
202	ADCC3	ADCC3 interrupt
203	ADCC4	ADCC4 interrupt
204	Reserved	Reserved
205	Reserved	Reserved
206	Reserved	Reserved
207	Reserved	Reserved
208	Reserved	Reserved
209	Reserved	Reserved
210	Reserved	Reserved
211	Reserved	Reserved
212	Reserved	Reserved
213	Reserved	Reserved
214	Reserved	Reserved
215	Reserved	Reserved
216	Reserved	Reserved
217	Reserved	RAM correctable error interrupt
218	FLASH_ECC_ERR	FLASH_ECC_ERR interrupt
219	Reserved	RAM access violation interrupt
220	SYS_PLL_SLIP	PLL slip interrupt
221	Reserved	Reserved
222	Reserved	CLA overflow interrupt
223	Reserved	CLA underflow interrupt
224	Reserved	-
225	CTI INT0	-
226	CTI INT1	-

3.4. Memory Mapping

The Flash access path of G32R501 has been optimized compared to its competitive product, and it supports both system bus access and ITCM interface access. G32R501 is equipped with two 128bit-wide Flash Banks, and can be configured as 128bit or 256bit access modes, improving the use flexibility. The following table shows the differences in Flash function between G32R501 and Txx320F28004x.

Table 21 Differences in Flash Function between G32R501 and Txx320F28004x

Function	G32R501	Txx320F28004x
Capacity	Maximum 640kB	Maximum 256kB
Memory bank	2 banks	2 banks
Flash memory waiting state	0 (SYSCLK ≤ 30) 1 (SYSCLK ≤ 60) 2 (SYSCLK ≤ 90) 3 (SYSCLK ≤120) 4 (SYSCLK ≤ 150) 5 (SYSCLK ≤ 180) 6 (SYSCLK ≤ 210) 7 (SYSCLK ≤ 240) 8 (SYSCLK ≤ 270)	0 (SYSCLK ≤ 20) 1 (SYSCLK ≤ 40) 2 (SYSCLK ≤ 60) 3 (SYSCLK ≤ 80) 4 (SYSCLK ≤ 100)
Sleep mode	/	Supported
Standby mode	Supported	Supported
Active mode	Supported	Supported
FACC acceleration	Supported	Supported
Busmatrix acceleration	Supported	Supported
Reading bit width	256 bits (single-bank mode)/128 bits (dual-bank mode)	128bit
Programming resolution	32bit	16bit
ECC	4 SECDED ECC	2 SECDED ECC
Erase	16kB (single-bank mode)/8kB (dual-bank mode)	8 kB

G32R501 supports 8-bit addressing, but accessing registers needs to follow the following rules:

Offset address: There are differences between the register and competitive product; please refer to User Manual or Table 26 for details.

Access rule: If the bit width of the register is 16 bits, G32R501 needs to be accessed according to 16 bits; if the bit width of the register is 32 bits, G32R501 needs to be accessed according to 32 bits.

The memory mapping of G32R501 is shown in the following table.

Table 22 Memory Mapping of G32R501

Address range (Hex)	Size (Max)	Description
0x0000_0000 – 0x0001_FFFF	128KB	CPU0 ITCM (64KB for single-core version, and 48KB for dual-core version by default)

Address range (Hex)	Size (Max)	Description
0xA000_0000 – 0xA001_FFFF	128KB	CPU0 ITCM (CPU0-AHBT) (64KB for single-core version, and 48KB for dual-core version by default)
0x0000_0000 – 0x0001_FFFF	128KB	CPU1 ITCM (0KB for single-core version, and 8KB for dual-core version by default)
0xA100_0000 – 0xA101_FFFF	128KB	CPU1 ITCM (CPU1-AHBT) (0KB for single-core version, and 8KB for dual-core version by default)
0x0008_0000 – 0x0008_BFFF	48KB	FLASH INFO on ITCM
0x0009_0000 – 0x0009_3FFF	16KB	FLASH INFO1 on ITCM
0x0010_0000 - 0x0019_FFFF	640KB	FLASH memory on ITCM
0x0800_0000 - 0x0809_FFFF	640KB	FLASH memory on BUSMATRIX
0x0810_0000 - 0x0810_BFFF	48KB	FLASH INFO on BUSMATRIX
0x0818_0000 - 0x0818_3FFF	16KB	FLASH INFO1 on BUSMATRIX
0x0900_0000 - 0x0901_FFFF	128KB	FLASH ECC
0x1000_0000 – 0x1001_FFFF	128KB	Boot ROM
0x1002_0000 – 0x1002_FFFF	64KB	Secure ROM
0x2000_0000 - 0x2001_FFFF	128KB	CPU0 DTCM (16KB for single-core version, and 16KB for dual-core version by default)
0xA010_0000 - 0xA011_FFFF	128KB	CPU0 DTCM (CPU0-AHBT) (16KB for single-core version, and 16KB for dual-core version by default)
0x2000_0000 – 0x2001_FFFF	128KB	CPU1 DTCM (0KB for single-core version, and 8KB for dual-core version by default)
0xA110_0000 - 0xA111_FFFF	128KB	CPU1 DTCM (CPU1-AHBT) (0KB for single-core version, and 8KB for dual-core version by default)
0x2010_0000 - 0x2011_FFFF	128KB	SRAM1 (Default 8KB)
0x2020_0000 - 0x2021_FFFF	128KB	SRAM2 (Default 8KB)
0x2030_0000 - 0x2031_FFFF	128KB	SRAM3 (Default 32KB)

The address bus allocation is as follows:

Table 23 Address Bus Allocation

Address range (Hex)	Bus	Function
0x4000_0000 – 0x4000_FFFF	APB0	APB peripherals
0x4001_0000 – 0x4001_FFFF	APB1	APB peripherals

0x4002_0000 – 0x4002_FFFF	APB2	APB peripherals
0x4003_0000 – 0x4003_FFFF	DEMUX0	AHB peripherals
0x5000_0000 – 0x5000_FFFF	APB3	APB peripherals
0x5000_0000 – 0x5002_FFFF	DEMUX1	AHB peripherals
0x6000_0000 – 0x6FFF_FFFF	-	-
0x5010_0000 – 0x5010_3FFF	APB4	APB peripherals
0x5010_4000 – 0x5010_FFFF	APB5	APB peripherals
0x5010_0000 – 0x5011_FFFF	DEMUX2	AHB peripherals

3.4.1. FLASH mapping

Up to two Flash memory banks (128KB for Bank0 and 512KB for Bank1) can be used on the G32R501x device. The Flash memory banks are controlled by two NVMC (nonvolatile memory controllers). Both CPU0 and CPU1 can access FLASH MEM through ITCM/C-BUS. The physical storage space accessed by two paths is the same, but the address is different. All FLASH erase/write operations are performed by operating the NVMC register. The FLASH MEM area address is read-only for CPU0/1 and DMA, and all write operations will be ignored. Any type of access should not be made to the Flash memory banks that are performing erasing/programming operations.

Table 24 Flash Sector Address (configured according to memory single bank (256 bits of reading width))

IP	Name	Sector base on C-BUS interface	Sector base on ITCM interface	Sector size	Bank number
Main memory	Sector0	0x0800_0000~0x0800_3FFF	0x0010_0000~0x0010_3FFF	16KByte	Bank0 & Bank1 mix
	Sector1	0x0800_4000~0x0800_7FFF	0x0010_4000~0x0010_7FFF	16KByte	Bank0 & Bank1 mix
	Sector2	0x0800_8000~0x0800_BFFF	0x0010_8000~0x0010_BFFF	16KByte	Bank0 & Bank1 mix
	……………	……………	……………	……………	Bank0 & Bank1 mix
	Sector14	0x0803_8000~0x0803_BFFF	0x0013_8000~0x0013_BFFF	16KByte	Bank0 & Bank1 mix
	Sector15	0x0803_C000~0x0803_FFFF	0x0013_C000~0x0013_FFFF	16KByte	Bank0 & Bank1 mix
	Sector16	0x0804_0000~0x0804_1FFF	0x0014_0000~0x0014_1FFF	8KByte	Bank1
	……………	……………	……………	……………	Bank1
	Sector63	0x0809_E000~0x0809_FFFF	0x0019_E000~0x0019_FFFF	8KByte	Bank1

IP	Name	Sector base on C-BUS interface	Sector base on ITCM interface	Sector size	Bank number
IFREN	Bank0 back-up sector0	0x0810_0000~0x0810_1FFF	0x0008_0000~0x0008_1FFF	8Kbyte	Bank0
	Bank0 back-up sector1	0x0810_2000~0x0810_3FFF	0x0008_2000~0x0008_3FFF	8Kbyte	Bank0
	Bank0 back-up sector2	0x0810_4000~0x0810_5FFF	0x0008_4000~0x0008_5FFF	8KByte	Bank0
	Bank0 back-up sector3	0x0810_6000~0x0810_7FFF	0x0008_6000~0x0008_7FFF	8KByte	Bank0
	Bank1 User Option byte	0x0810_8000~0x0810_9FFF	0x0008_8000~0x0008_9FFF	8Kbyte	Bank1
	Bank1 OTP	0x0810_A000~0x0810_BFFF	0x0008_A000~0x0008_BFFF	8Kbyte	Bank1
IFREN1	FLASH trimming	0x0818_0000~0x0818_3FFF	0x0009_0000~0x0009_3FFF	16KByte	Bank0 & Bank1 mix
	ECC	0x0900_0000~0x0902_17FF	NA	128Kbyte (half-word access only)	Bank0 & Bank1 mix

Table 25 Flash Sector Address (configured according to memory dual bank (128 bits of reading width))

IP	Name	Sector base on C-BUS interface	Sector base on ITCM interface	Sector size	Bank number
Main memory bank0	Sector0	0x0800_0000~0x0800_1FFF	0x0010_0000~0x0010_1FFF	8KByte	Bank0
	Sector1	0x0800_2000~0x0800_3FFF	0x0010_2000~0x0010_3FFF	8KByte	Bank0
	Sector2	0x0800_4000~0x0800_5FFF	0x0010_4000~0x0010_5FFF	8KByte	Bank0
	……………	……………	……………	……………	Bank0

IP	Name	Sector base on C-BUS interface	Sector base on ITCM interface	Sector size	Bank number
	Sector14	0x0801_C000~0x0801_DFFF	0x0011_C000~0x0011_DFFF	8KByte	Bank0
	Sector15	0x0801_E000~0x0801_FFFF	0x0011_E000~0x0011_FFFF	8KByte	Bank0
Main memory bank1	Sector16	0x0802_0000~0x0802_1FFF	0x0012_0000~0x0012_1FFF	8KByte	Bank1
	Sector17	0x0802_2000~0x0802_3FFF	0x0012_2000~0x0012_3FFF	8KByte	Bank1
	Sector18	0x0802_4000~0x0802_5FFF	0x0012_4000~0x0012_5FFF	8Kbyte1	Bank1
	……………	……………	……………	……………	Bank1
	Sector30	0x0803_C000~0x0803_DFFF	0x0013_C000~0x0013_DFFF	8KByte	Bank1
	Sector31	0x0803_E000~0x0803_FFFF	0x0013_E000~0x0013_FFFF	8KByte	Bank1
	Sector32	0x0804_0000~0x0804_1FFF	0x0014_0000~0x0014_1FFF	8KByte	Bank1
	……………	……………	……………	……………	Bank1
	Sector79	0x0809_E000~0x0809_FFFF	0x0019_E000~0x0019_FFFF	8KByte	Bank1
IFREN	Bank0 back-up sector0	0x0810_0000~0x0810_1FFF	0x0008_0000~0x0008_1FFF	8Kbyte	Bank0
	Bank0 back-up sector1	0x0810_2000~0x0810_3FFF	0x0008_2000~0x0008_3FFF	8Kbyte	Bank0
	Bank0 back-up sector2	0x0810_4000~0x0810_5FFF	0x0008_4000~0x0008_5FFF	8KByte	Bank0
	Bank0 back-up sector3	0x0810_6000~0x0810_7FFF	0x0008_6000~0x0008_7FFF	8KByte	Bank0
	Bank1 User Option byte	0x0810_8000~0x0810_9FFF	0x0008_8000~0x0008_9FFF	8Kbyte	Bank1
	Bank1 OTP	0x0810_A000~0x0810_BFFF	0x0008_A000~0x0008_BFFF	8Kbyte	Bank1
IFREN1	FLASH trimming	0x0818_0000~0x0818_3FFF	0x0009_0000~0x0009_3FFF	16KByte	Bank0 & Bank1 mix
	ECC	0x0900_0000~0x0902_17FF	NA	128Kbyte (half-word access only)	Bank0 & Bank1 mix

3.4.2. Peripheral register memory mapping

Table 26 Peripheral Register Memory Mapping

Address	Bus	IP
0x4000_0000-0x4000_03FF	APB0	PWM1
0x4000_0400-0x4000_07FF		PWM2
0x4000_0800-0x4000_0BFF		PWM3
0x4000_0C00-0x4000_0FFF		PWM4
0x4000_1000-0x4000_13FF		PWM5
0x4000_1400-0x4000_17FF		PWM6
0x4000_1800-0x4000_1BFF		PWM7
0x4000_1C00-0x4000_1FFF		PWM8
0x4000_2000-0x4000_23FF		CAP1
0x4000_2400-0x4000_27FF		CAP2
0x4000_2800-0x4000_2BFF		CAP3
0x4000_2C00-0x4000_2FFF		CAP4
0x4000_3000-0x4000_33FF		CAP5
0x4000_3400-0x4000_37FF		CAP6
0x4000_3800-0x4000_3BFF		CAP7
0x4000_3C00-0x4000_3FFF		SDF
0x4000_4000-0x4000_FFFF		Reserved
0x4001_1C00-0x4001_1FFF	APB1	COMP1
0x4001_2000-0x4001_23FF		COMP2
0x4001_2400-0x4001_27FF		COMP3
0x4001_2800-0x4001_2BFF		COMP4
0x4001_2C00-0x4001_2FFF		COMP5
0x4001_3000-0x4001_33FF		COMP6
0x4001_3400-0x4001_37FF		COMP7
0x4001_3800-0x4001_3BFF		QEP1
0x4001_3C00-0x4001_3FFF		QEP2
0x4001_4000-0x4001_FFFF		Reserved
0x4002_0000-0x4002_03FF	APB2	ADCA
0x4002_0400-0x4002_07FF		ADCB
0x4002_0800-0x4002_0BFF		ADCC

Address	Bus	IP
0x4002_0C00-0x4002_FFFF		Reserved
0x4003_0000-0x4003_07FF	DEMUX0	GPIOCTRL
0x4003_0800-0x4003_0BFF		GPIODATA
0x4003_0C00-0x4003_0C7F		INPUTXBAR
0x4003_0C80-0x4003_0FFF		SyncSocREG
0x4003_1000-0x4003_103F		XBAR_REG
0x4003_11C0-0x4003_123F		PWM_XBAR_REG
0x4003_1240-0x4003_12BF		FLB_XBAR_REG
0x4003_12C0-0x4003_133F		OUTPUT_XBAR_REG
0x4003_1400-0x4003_FFFF		Reserved
0x5000_0000-0x5000_03FF	APB3	LIN
0x5000_0400-0x5000_0BFF		Reserved
0x5000_0C00-0x5000_0FFF		UARTA
0x5000_1000-0x5000_13FF		SPIA
0x5000_1400-0x5000_17FF		I2C
0x5000_1800-0x5000_1BFF		DACA
0x5000_1C00-0x5000_1FFF		DACB
0x5000_2000-0x5000_27FF		CANA
0x5000_2800-0x5000_FFFF		Reserved
0x5000_0000-0x5000_FFFF	DEMUX1	APB3
0x5001_0000-0x5001_07FF		NVMC
0x5001_0800-0x5001_0BFF		CFGSMS
0x5001_0C00-0x5001_FFFF		Reserved
0x5002_0000-0x5002_3FFF		SYSC
0x5002_4000-0x5002_5FFF		DCS
0x5002_6000-0x5002_63FF		QSPI
0x5002_6400-0x5002_64BF		WWDT
0x5002_64C0-0x5002_67FF		NMIWDT
0x5002_6800-0x5002_7FFF		Reserved
0x5002_8000-0x5002_83FF		ANALOG_SUB_SYSTEM
0x5002_8400-0x5002_FFFF		Reserved
0x6000_0000-0x6FFF_FFFF		QSPI_MEMORY

Address	Bus	IP
0x5010_0000-0x5010_03FF	APB4	UARTB
0x5010_0400-0x5010_07FF		SPIB
0x5010_0800-0x5010_0BFF		PMBUS
0x5010_0C00-0x5010_0FFF		Reserved
0x5010_1000-0x5010_13FF		Reserved
0x5010_1400-0x5010_17FF		DCC
0x5010_1800-0x5010_1FFF		CANB
0x5010_2000-0x5010_27FF		FLB1
0x5010_2800-0x5010_2FFF		FLB2
0x5010_3000-0x5010_37FF		FLB3
0x5010_3800-0x5010_3FFF		FLB4
0x5010_0000-0x5010_3FFF	DEMUX2	APB4
0x5010_4000-0x5010_FFFF		APB5
0x5011_0000-0x5011_03FF		TMR0
0x5011_0400-0x5011_07FF		TMR1
0x5011_0800-0x5011_0BFF		TMR2
0x5011_0C00-0x5011_0FFF		DMA
0x5011_1000-0x5011_13FF		EXTI
0x5011_1400-0x5011_FFFF		Reserved

3.5. Clock Tree Differences

Use of G32R501 clock is similar to that of Txx320F28004x, supporting INTOSC1, INTOSC2, XTAL, and PLL. The internal PLL output frequency range of G32R501 is larger, supporting up to 300MHz clock output (PLLRAWCLK). Besides, because it adopts dual-core architecture and the operating frequency of the CPU and CPU subsystems is higher than peripherals (the maximum operating frequency of peripherals is 125MHz), the extra clock control part is controlled by the newly added clock control unit module.

APB clock division and dual-core system clock control are added to the clock control unit module in the clock control logic of Txx320F28004x. For more detailed information about clock structure, please refer to the *G32R501 User Manual*.

Table 27 Clock Structure Comparison

Clock	G32R501	Txx320F28004x
OSCCLK	Supported	Supported
PLLRAWCLK	Supported	Supported

Clock	G32R501	Txx320F28004x
WDCLK	Supported	Supported
PLLSYSCLK	Supported	Supported
SYSCLK	Supported	Supported
LSPCLK	Supported	Supported
APBCLK	Supported	Not supported
CPU0CLK	Supported	Supported
CPU1CLK	Supported	Not supported

3.6. Differences in Peripherals

The DMA of Txx320F28004x does not support SCI/I2C/Flash data transmission, while the DMA of G32R501 supports data transmission of all bus peripherals.

4. Using the Library for Firmware Porting

During the migration of Txx320F28004x series microcontrollers to G32R501 series microcontrollers, the porting of firmware library is a critical step. The firmware library not only provides abstraction and package of hardware peripherals, but also simplifies the development process, and improves the code maintainability and portability. The firmware library of different microcontroller series may have significant differences in API design, naming rules, and implementation methods. Therefore, it is necessary to conduct detailed analysis and adaptation on the existing firmware library in the migration process.

This chapter will introduce how to migrate the firmware library of Txx320F28004x series to G32R501 series. We will focus on discussing the compatibility between the bit field library and the driver library, and how to implement the existing functions on new platforms. In addition, some optimization and testing suggestions will be provided to ensure that the migrated firmware library can run efficiently.

For more detailed content of API situation and migration references involved in the G32R501 series microcontrollers, please refer to the following tables:

- G32R501 device_support (Bit Field Library) API Migration Reference Table.xlsx
- G32R501 driverlib (Firmware Library) API Migration Reference Table.xlsx

4.1. Bit Field Library Porting

The bit field library is used in the embedded system to simplify the access and operation of hardware registers.

There are certain differences in register naming and bit field library support between G32R501 and Txx320F28004x. In the porting process, it is necessary to refer to the G32R501 User Manual to modify register naming one by one in order to ensure that all register operations can be correctly mapped to the new hardware platform.

This section will provide a detailed introduction on how to port the bit field library of Txx320F28004x to G32R501, and the key matters needing to be noticed in this process. We will discuss in detail the differences in register naming.

Note: For brevity, the naming difference of registers in this section will not be accurate to the naming differences of bit fields. This difference requires users to refer to the *G32R501 User Manual* themselves.

The general steps for porting a bit field library are summarized as follows:

- Analyze the existing code:
  - Browse and understand the bit field library used in existing code, and record the peripherals and registers involved.
- Compare the register naming:
  - Refer to the *G32R501 Hardware Manual*, find the corresponding register name and

record it.

- Modify the code:
  - Replace the Txx320F28004x register name used in the existing code with the register name of G32R501.
  - Ensure that the bit field definition matches the new register layout.

4.1.1. ADC module

The ADC module of G32R501 is compatible with the ADC module of Txx320F28004x. In terms of register naming, the two are the same.

4.1.2. AS module/ANALOGSUBSYS module

The AS module of G32R501 is compatible with the ANALOGSUBSYS module of Txx320F28004x. In terms of register naming, the two are not the same. Please refer to the comparison in Table 28 during use.

Table 28 Register Naming Comparison of AS Module

G32R501	Txx320F28004x
AS_REGS	**ANALOG_SUBSYS_REGS**
ANAREFPP	ANAREFPP
TSNSCTL	TSNSCTL
ANAREFCTL	ANAREFCTL
VMONCTL	VMONCTL
/	DCDCCTL
/	DCDCSTS
CMPHPMXSEL	CMPHPMXSEL
CMPLPMXSEL	CMPLPMXSEL
CMPHNMXSEL	CMPHNMXSEL
CMPLNMXSEL	CMPLNMXSEL
ADCDACLOOPBACK	/
LOCK	LOCK

4.1.3. CAN module

The CAN module of G32R501 is compatible with the CAN module of Txx320F28004x. In terms of register naming, the two are not the same. Please refer to the comparison in Table 29 during use.

Table 29 Register Naming Comparison of CAN Module

G32R501	Txx320F28004x
CAN_REGS	**CAN_REGS**
CAN_CTRL	CAN_CTL
CAN_EFLG	CAN_ES
CAN_ECNT	CAN_ERRC
CAN_BTIM	CAN_BTR
CAN_INT	CAN_INT
CAN_TEST	CAN_TEST
CAN_PE	CAN_PERR
CAN_RAMINIT	CAN_RAM_INIT
CAN_GLBINTEN	CAN_GLB_INT_EN
CAN_GLBINTFLG	CAN_GLB_INT_FLG
CAN_GLBINTCLR	CAN_GLB_INT_CLR
CAN_ABOTR	CAN_ABOTR
CAN_TXREQFLG	CAN_TXRQ_X
CAN_TXRQ_21	CAN_TXRQ_21
CAN_NDATAFLG	CAN_NDAT_X
CAN_NDAT_21	CAN_NDAT_21
CAN_IPENDFLG	CAN_IPEN_X
CAN_IPEN_21	CAN_IPEN_21
CAN_MVALFLG	CAN_MVAL_X
CAN_MVAL_21	CAN_MVAL_21
CAN_IP_MUX21	CAN_IP_MUX21
CAN_IF1COM	CAN_IF1CMD
CAN_IF1MASK	CAN_IF1MSK
CAN_IF1ARB	CAN_IF1ARB
CAN_IF1MCTRL	CAN_IF1MCTL
CAN_IF1DATAA	CAN_IF1DATA
CAN_IF1DATAB	CAN_IF1DATB
CAN_IF2COM	CAN_IF2CMD
CAN_IF2MASK	CAN_IF2MSK
CAN_IF2ARB	CAN_IF2ARB

G32R501	Txx320F28004x
CAN_REGS	**CAN_REGS**
CAN_IF2MCTRL	CAN_IF2MCTL
CAN_IF2DATAA	CAN_IF2DATA
CAN_IF2DATAB	CAN_IF2DATB
CAN_IF3OBS	CAN_IF3OBS
CAN_IF3MASK	CAN_IF3MSK
CAN_IF3ARB	CAN_IF3ARB
CAN_IF3MCTRL	CAN_IF3MCTL
CAN_IF3DATAA	CAN_IF3DATA
CAN_IF3DATAB	CAN_IF3DATB
CAN_IF3UPD	CAN_IF3UPD

4.1.4. CLA module

G32R501 does not have this module.

4.1.5. FLB module/CLB module

The FLB module of G32R501 is compatible with the CLB module of Txx320F28004x. In terms of register naming, the two are not the same. Please refer to the comparison in Table 30, Table 31 and Table 32 during use.

Table 30 LOGIC_CONFIG Register Naming Comparison of FLB Module

G32R501	Txx320F28004x
FLB_LOGIC_CONFIG_REGS	**CLB_LOGIC_CONFIG_REGS**
FLB_COUNT_RESET	CLB_COUNT_RESET
FLB_COUNT_MODE_1	CLB_COUNT_MODE_1
FLB_COUNT_MODE_0	CLB_COUNT_MODE_0
FLB_COUNT_EVENT	CLB_COUNT_EVENT
FLB_FSM_EXTRA_IN0	CLB_FSM_EXTRA_IN0
FLB_FSM_EXTERNAL_IN0	CLB_FSM_EXTERNAL_IN0
FLB_FSM_EXTERNAL_IN1	CLB_FSM_EXTERNAL_IN1
FLB_FSM_EXTRA_IN1	CLB_FSM_EXTRA_IN1
FLB_LUT4_IN0	CLB_LUT4_IN0
FLB_LUT4_IN1	CLB_LUT4_IN1

G32R501	Txx320F28004x
FLB_LOGIC_CONFIG_REGS	**CLB_LOGIC_CONFIG_REGS**
FLB_LUT4_IN2	CLB_LUT4_IN2
FLB_LUT4_IN3	CLB_LUT4_IN3
FLB_FSM_LUT_FN1_0	CLB_FSM_LUT_FN1_0
FLB_FSM_LUT_FN2	CLB_FSM_LUT_FN2
FLB_LUT4_FN1_0	CLB_LUT4_FN1_0
FLB_LUT4_FN2	CLB_LUT4_FN2
FLB_FSM_NEXT_STATE_0	CLB_FSM_NEXT_STATE_0
FLB_FSM_NEXT_STATE_1	CLB_FSM_NEXT_STATE_1
FLB_FSM_NEXT_STATE_2	CLB_FSM_NEXT_STATE_2
FLB_MISC_CONTROL	CLB_MISC_CONTROL
FLB_OUTPUT_LUT_0	CLB_OUTPUT_LUT_0
FLB_OUTPUT_LUT_1	CLB_OUTPUT_LUT_1
FLB_OUTPUT_LUT_2	CLB_OUTPUT_LUT_2
FLB_OUTPUT_LUT_3	CLB_OUTPUT_LUT_3
FLB_OUTPUT_LUT_4	CLB_OUTPUT_LUT_4
FLB_OUTPUT_LUT_5	CLB_OUTPUT_LUT_5
FLB_OUTPUT_LUT_6	CLB_OUTPUT_LUT_6
FLB_OUTPUT_LUT_7	CLB_OUTPUT_LUT_7
FLB_HLC_EVENT_SEL	CLB_HLC_EVENT_SEL
FLB_COUNT_MATCH_TAP_SEL	CLB_COUNT_MATCH_TAP_SEL
FLB_OUTPUT_COND_CTRL_0	CLB_OUTPUT_COND_CTRL_0
FLB_OUTPUT_COND_CTRL_1	CLB_OUTPUT_COND_CTRL_1
FLB_OUTPUT_COND_CTRL_2	CLB_OUTPUT_COND_CTRL_2
FLB_OUTPUT_COND_CTRL_3	CLB_OUTPUT_COND_CTRL_3
FLB_OUTPUT_COND_CTRL_4	CLB_OUTPUT_COND_CTRL_4
FLB_OUTPUT_COND_CTRL_5	CLB_OUTPUT_COND_CTRL_5
FLB_OUTPUT_COND_CTRL_6	CLB_OUTPUT_COND_CTRL_6
FLB_OUTPUT_COND_CTRL_7	CLB_OUTPUT_COND_CTRL_7

Table 31 LOGIC_ CONTROL Register Naming Comparison of FLB Module

G32R501	Txx320F28004x
FLB_LOGIC_CONTROL_REGS	**CLB_LOGIC_CONTROL_REGS**
FLB_LOAD_EN	CLB_LOAD_EN
FLB_LOAD_ADDR	CLB_LOAD_ADDR
FLB_LOAD_DATA	CLB_LOAD_DATA
FLB_INPUT_FILTER	CLB_INPUT_FILTER
FLB_IN_MUX_SEL_0	CLB_IN_MUX_SEL_0
FLB_LCL_MUX_SEL_1	CLB_LCL_MUX_SEL_1
FLB_LCL_MUX_SEL_2	CLB_LCL_MUX_SEL_2
FLB_BUF_PTR	CLB_BUF_PTR
FLB_GP_REG	CLB_GP_REG
FLB_OUT_EN	CLB_OUT_EN
FLB_GLBL_MUX_SEL_1	CLB_GLBL_MUX_SEL_1
FLB_GLBL_MUX_SEL_2	CLB_GLBL_MUX_SEL_2
FLB_PRESCALE_CTRL	CLB_PRESCALE_CTRL
FLB_INTR_TAG_REG	CLB_INTR_TAG_REG
FLB_LOCK	CLB_LOCK
FLB_DBG_OUT_2	CLB_DBG_OUT_2
FLB_DBG_R0	CLB_DBG_R0
FLB_DBG_R1	CLB_DBG_R1
FLB_DBG_R2	CLB_DBG_R2
FLB_DBG_R3	CLB_DBG_R3
FLB_DBG_C0	CLB_DBG_C0
FLB_DBG_C1	CLB_DBG_C1
FLB_DBG_C2	CLB_DBG_C2
FLB_DBG_OUT	CLB_DBG_OUT

Table 32 DATA_EXCHANGE Register Naming Comparison of FLB Module

G32R501	Txx320F28004x
FLB_DATA_EXCHANGE_REGS	**CLB_DATA_EXCHANGE_REGS**
FLB_PUSH[4]	CLB_PUSH[4]
FLB_PULL[4]	CLB_PULL[4]

4.1.6. FLBXBAR module/CLBXBAR module

The FLBXBAR module of G32R501 is compatible with the CLBXBAR module of Txx320F28004x. In terms of register naming, the two are the same.

4.1.7. COMP module/CMPSS module

The COMP module of G32R501 is compatible with the CMPSS module of Txx320F28004x. In terms of register naming, the two are the same.

4.1.8. TMR module/CPUTIMER module

The TMR module of G32R501 is compatible with the CPUTIMER module of Txx320F28004x. In terms of register naming, the two are the same.

4.1.9. DAC module

The DAC module of G32R501 is compatible with the DAC module of Txx320F28004x. In terms of register naming, the two are the same.

4.1.10. DCCOMP module/DCC module

The DCCOMP module of G32R501 is compatible with the DCC module of Txx320F28004x. In terms of register naming, the two are not the same. Please refer to the comparison in Table 33 during use.

Table 33 Register Naming Comparison of DCCOMP Module

G32R501	Txx320F28004x
DCCOMP_REGS	**DCC_REGS**
DCCOMPCTRL	DCCGCTRL
DCCOMPREV	DCCREV
DCCOMPCNTSEED0	DCCCNTSEED0
DCCOMPVALSEED0	DCCVALIDSEED0
DCCOMPCNTSEED1	DCCCNTSEED1
DCCOMPFLG	DCCSTATUS
DCCOMPCLKSRCCNT0	DCCCNT0
DCCOMPCLKSRCVAL0	DCCVALID0
DCCOMPCLKSRCCNT1	DCCCNT1
DCCOMPCLKSRCSEL1	DCCCLKSRC1

DCCOMPCLKSRCSEL0	DCCCLKSRC0

4.1.11. DCS module/DCSM module

The DCS module of G32R501 is compatible with the DCSM module of Txx320F28004x. In terms of register naming, the two are not the same. Please refer to the comparison in Table 34, Table 35, Table 36, Table 37 and Table 38 during use.

Table 34 B0Z1 Register Naming Comparison of DCS Module

G32R501	Txx320F28004x
DCS_B0Z1_REGS	**DCSM_BANK0_Z1_REGS**
B0Z1LP	B0_Z1_LINKPOINTER
Z1LOCK	Z1_OTPSECLOCK
Z1BOOTH	Z1_BOOTDEF_HIGH
B0Z1LPERR	B0_Z1_LINKPOINTERERR
Z1BOOTPINCFG	Z1_BOOTPIN_CONFIG
Z1GENPUR	Z1_GPREG2
Z1BOOTL	Z1_BOOTDEF_LOW
Z1CSKEY0	Z1_CSMKEY0
Z1CSKEY1	Z1_CSMKEY1
Z1CSKEY2	Z1_CSMKEY2
Z1CSKEY3	Z1_CSMKEY3
Z1CSCSTS	Z1_CR
B0Z1GRABSECT	B0_Z1_GRABSECTR
Z1GRABRAM	Z1_GRABRAMR
B0Z1EXESECT	B0_Z1_EXEONLYSECTR
Z1EXERAM	Z1_EXEONLYRAMR

Table 35 B0Z2 Register Naming Comparison of DCS Module

G32R501	Txx320F28004x
DCS_B0Z2_REGS	**DCSM_BANK0_Z2_REGS**
B0Z2LP	B0_Z2_LINKPOINTER
Z2LOCK	Z2_OTPSECLOCK
B0Z2LPERR	B0_Z2_LINKPOINTERERR
Z2CSKEY0	Z2_CSMKEY0
Z2CSKEY1	Z2_CSMKEY1

G32R501	Txx320F28004x
DCS_B0Z2_REGS	**DCSM_BANK0_Z2_REGS**
Z2CSKEY2	Z2_CSMKEY2
Z2CSKEY3	Z2_CSMKEY3
Z2CSCSTS	Z2_CR
B0Z2GRABSECT	B0_Z2_GRABSECTR
Z2GRABRAM	Z2_GRABRAMR
B0Z2EXESECT	B0_Z2_EXEONLYSECTR
Z2EXERAM	Z2_EXEONLYRAMR

Table 36 COMMON Register Naming Comparison of DCS Module

G32R501	Txx320F28004x
DCS_COMMON_REGS	**DCSM_COMMON_REGS**
FLSE	FLSEM
B0SECT	B0_SECTSTAT
RAM	RAMSTAT
B1SECT	B1_SECTSTAT
SEERR	SECERRSTAT
SEERRCLR	SECERRCLR
SEERRSET	SECERRFRC

Table 37 B1Z1 Register Naming Comparison of DCS Module

G32R501	Txx320F28004x
DCS_B1Z1_REGS	**DCSM_BANK1_Z1_REGS**
B1Z1LP	B1_Z1_LINKPOINTER
B1Z1LPERR	B1_Z1_LINKPOINTERERR
B1Z1GRABSECT	B1_Z1_GRABSECTR
B1Z1EXESECT	B1_Z1_EXEONLYSECTR

Table 38 B1Z2 Register Naming Comparison of DCS Module

G32R501	Txx320F28004x
DCS_B1Z2_REGS	**DCSM_BANK1_Z2_REGS**
B1Z2LP	B1_Z2_LINKPOINTER
B1Z2LPERR	B1_Z2_LINKPOINTERERR

B2Z2GRABSECT	B1_Z2_GRABSECTR
B1Z2EXESECT	B1_Z2_EXEONLYSECTR

4.1.12. DMA module

The DMA module of G32R501 is compatible with the DMA module of Txx320F28004x. In terms of register naming, the two are the same.

4.1.13. CAP module/ECAP module

The CAP module of G32R501 is compatible with the ECAP module of Txx320F28004x. In terms of register naming, the two are not the same. Please refer to the comparison in Table 39 during use.

Table 39 Register Naming Comparison of CAP Module

G32R501	Txx320F28004x
CAP_REGS	**ECAP_REGS**
CCTL0	ECCTL0
CCTL1	ECCTL1
CCTL2	ECCTL2
CEINT	ECEINT
CFLG	ECFLG
CCLR	ECCLR
CFRC	ECFRC

4.1.14. PWM module/EPWM module

The PWM module of G32R501 is compatible with the EPWM module of Txx320F28004x. In terms of register naming, the two are not the same. Please refer to the comparison in Table 40 during use.

In the simulation environment, if the user needs to set a breakpoint inside the PWM interrupt function, the user must configure the breakpoint to automatically stop the counter, so that the counter does not continue counting when the simulation is paused. Otherwise, new interrupts may become "blocked," causing the false impression that the interrupt is unresponsive. When initializing the PWM, enable the simulation stop signal for the peripheral.

Table 40 Register Naming Comparison of PWM Module

G32R501	Txx320F28004x
PWM_REGS	**EPWM_REGS**

PWMXLINK	EPWMXLINK
PWMLOCK	EPWMLOCK

4.1.15. PWM_XBAR module/EPWM_XBAR module

The PWM_XBAR module of G32R501 is compatible with the EPWM_XBAR module of Txx320F28004x. In terms of register naming, the two are the same.

4.1.16. QEP module/EQEP module

The QEP module of G32R501 is compatible with the EQEP module of Txx320F28004x. In terms of register naming, the two are the same.

4.1.17. ERAD module

G32R501 does not have this module.

4.1.18. FLASH module

There are significant differences between the FLASH module of G32R501 and the FLASH module of Txx320F28004x. Please refer to the relevant manual for the specific usage.

4.1.19. GPIO module

The GPIO module of G32R501 is compatible with the GPIO module of Txx320F28004x. In terms of register naming, the two are the same.

But the GPIO module of G32R501 has several additional registers. Please refer to the relevant manual and the comparison in Table 41 for specific usage.

Table 41 Differences of GPIO Module Register

G32R501	Txx320F28004x
GPIO_CTRL_REGS	**GPIO_CTRL_REGS**
GPADSEL1	/
GPADSEL2	/
GPAFEN	/
GPBDSEL1	/
GPBDSEL2	/
GPBFEN	/

4.1.20. I2C module

The I2C module of G32R501 is compatible with the I2C module of Txx320F28004x. In terms of register naming, the two are not the same. Please refer to the comparison in Table 42 during use.

Table 42 Register Naming Comparison of I2C Module

G32R501	**Txx320F28004x**
I2C_REGS	**I2C_REGS**
I2COADDR	I2COAR
I2CIREN	I2CIER
I2CSTS	I2CSTR
I2CCLKL	I2CCLKL
I2CCLKH	I2CCLKH
I2CCNT	I2CCNT
I2CRXD	I2CDRR
I2CSADDR	I2CSAR
I2CTXD	I2CDXR
I2CCTRL	I2CMDR
I2CISRC	I2CISRC
I2CEXT	I2CEMDR
I2CPSC	I2CPSC
I2CTXFIFO	I2CFFTX
I2CRXFIFO	I2CFFRX

4.1.21. INTPUT_XBAR module

The INTPUT_XBAR module of G32R501 is compatible with the INTPUT_XBAR module of Txx320F28004x. In terms of register naming, the two are the same.

4.1.22. LIN module

The LIN module of G32R501 is compatible with the LIN module of Txx320F28004x. In terms of register naming, the two are not the same. Please refer to the comparison in Table 43 during use.

Table 43 Register Naming Comparison of LIN Module

G32R501	Txx320F28004x
LIN_REGS	**LIN_REGS**
LINGCTRL0	SCIGCR0
LINGCTRL1	SCIGCR1
LINGCTRL2	SCIGCR2
LINIEN	SCISETINT
LINICLR	SCICLEARINT
LINILEN	SCISETINTLVL
LINILCLR	SCICLEARINTLVL
LINFLG	SCIFLR
LINIVO0	SCIINTVECT0
LINIVO1	SCIINTVECT1
LINLCFG	SCIFORMAT
LINBR	BRSR
LINRXED	SCIED
LINRXD	SCIRD
LINTXD	SCITD
LINPCTRL0	SCIPIO0
LINPCTRL2	SCIPIO2
LINCOMP	LINCOMP
LINRXB0	LINRD0
LINRXB1	LINRD1
LINIDMASK	LINMASK
LINID	LINID
LINTXB0	LINTD0
LINTXB1	LINTD1
LINMBRPSC	MBRSR
LINEET	IODFTCTRL
LINGIEN	LIN_GLB_INT_EN
LINGIFLG	LIN_GLB_INT_FLG
LINGICLR	LIN_GLB_INT_CLR

4.1.23. NMI module/NMIINTRUPT module

The NMI module of G32R501 is compatible with the NMIINTRUPT module of Txx320F28004x. In terms of register naming, the two are the same.

4.1.24. OUTPUT_XBAR module

The OUTPUT_XBAR module of G32R501 is compatible with the OUTPUT_XBAR module of Txx320F28004x. In terms of register naming, the two are the same.

4.1.25. PGA module

G32R501 does not have this module.

4.1.26. PIECTRL module

G32R501 does not have this module.

4.1.27. PMBUS module

The PMBUS module of G32R501 is compatible with the PMBUS module of Txx320F28004x. In terms of register naming, the two are not the same. Please refer to the comparison in Table 44 during use.

Table 44 Register Naming Comparison of PMBUS Module

G32R501	Txx320F28004x
PMBUS_REGS	**PMBUS_REGS**
PMBUS_MCTRL	PMBMC
PMBUS_TXB	PMBTXBUF
PMBUS_RXB	PMBRXBUF
PMBUS_ACK	PMBACK
PMBUS_STS	PMBSTS
PMBUS_IMASK	PMBINTM
PMBUS_SCTRL	PMBSC
PMBUS_HSADDR	PMBHSA
PMBUS_CTRL	PMBCTRL
PMBUS_TIMCTRL	PMBTIMCTL
PMBUS_CLKTIM	PMBTIMCLK
PMBUS_STATIM	PMBTIMSTSETUP

PMBUS_BUSIDTIM	PMBTIMBIDLE
PMBUS_CLKLTOTIIM	PMBTIMLOWTIMOUT
PMBUS_CLKHTOTIIM	PMBTIMHIGHTIMOUT

4.1.28. UART module/SCI module

The UART module of G32R501 is compatible with the SCI module of Txx320F28004x. In terms of register naming, the two are not the same. Please refer to the comparison in Table 45 during use.

During the UART initialization process, if the RX-related IO pin is floating, invalid data may be received after initialization is completed. This is because the GPIO is not configured with pull-up or pull-down resistors, causing the signal level from the IO pin to the internal digital circuit to be unpredictable (related to the analog electrical characteristics). Therefore, configuring the RX GPIO pin with a pull-up resistor during initialization can prevent this issue.

Table 45 Register Naming Comparison of UART Module

G32R501	**Txx320F28004x**
UART_REGS	**SCI_REGS**
UARTCCR	SCICCR
UARTCTL1	SCICTL1
UARTHBAUD	SCIHBAUD
UARTLBAUD	SCILBAUD
UARTCTL2	SCICTL2
UARTRXST	SCIRXST
UARTRXEMU	SCIRXEMU
UARTRXBUF	SCIRXBUF
UARTTXBUF	SCITXBUF
UARTFFTX	SCIFFTX
UARTFFRX	SCIFFRX
UARTFFCT	SCIFFCT
UARTPRI	SCIPRI

4.1.29. SDF module/SDFM module

The SDF module of G32R501 is compatible with the SDFM module of Txx320F28004x. In terms of register naming, the two are the same.

4.1.30. SPI module

The SPI module of G32R501 is compatible with the SPI module of Txx320F28004x. In terms of register naming, the two are not the same. Please refer to the comparison in Table 46 during use.

Table 46 Register Naming Comparison of SPI Module

G32R501	Txx320F28004x
SPI_REGS	**SPI_REGS**
SPICFG	SPICCR
SPICTRL	SPICTL
SPISTS	SPISTS
SPIBR	SPIBRR
SPIRXEMU	SPIRXEMU
SPIRXBUF	SPIRXBUF
SPITXBUF	SPITXBUF
SPIDAT	SPIDAT
SPITXFIFO	SPIFFTX
SPIRXFIFO	SPIFFRX
SPIFFCTRL	SPIFFCT
SPIPRICTRL	SPIPRI

4.1.31. SYSCTRL module

The SYSCTRL module of G32R501 is compatible with the SYSCTRL module of Txx320F28004x. In terms of register naming, the two are not the same. There are also differences in registers. Please refer to the relevant documents and the comparison in Table 47, Table 48, Table 49 and Table 50 during use.

Table 47 DEV Register Naming Comparison of SYSCTRL Module

G32R501	Txx320F28004x
DEV_REGS	**DEV_CFG_REGS**
PARTIDL	PARTIDL
PARTIDH	PARTIDH
REVID	REVID
/	DC21
/	FUSEERR

/	SOFTPRES0
SOFTPRES2	SOFTPRES2
SOFTPRES3	SOFTPRES3
SOFTPRES4	SOFTPRES4
SOFTPRES6	SOFTPRES6
SOFTPRES7	SOFTPRES7
SOFTPRES8	SOFTPRES8
SOFTPRES9	SOFTPRES9
SOFTPRES10	SOFTPRES10
SOFTPRES11	/
SOFTPRES13	SOFTPRES13
SOFTPRES14	SOFTPRES14
/	SOFTPRES15
SOFTPRES16	SOFTPRES16
SOFTPRES17	SOFTPRES17
SOFTPRES19	SOFTPRES19
SOFTPRES20	SOFTPRES20
/	SOFTPRES40
PACKSEL	TAP_STATUS

Table 48 CPU Register Naming Comparison of SYSCTRL Module

G32R501	**Txx320F28004x**
CPU_REGS	**CPU_SYS_REGS**
CPUSYSLOCK1	CPUSYSLOCK1
/	PIEVERRADDR
PCLKCR0	PCLKCR0
PCLKCR1	/
PCLKCR2	PCLKCR2
PCLKCR3	PCLKCR3
PCLKCR4	PCLKCR4
PCLKCR6	PCLKCR6
PCLKCR7	PCLKCR7
PCLKCR8	PCLKCR8
PCLKCR9	PCLKCR9

PCLKCR10	PCLKCR10
PCLKCR13	PCLKCR13
PCLKCR14	PCLKCR14
/	PCLKCR15
PCLKCR16	PCLKCR16
PCLKCR17	PCLKCR17
PCLKCR18	PCLKCR18
PCLKCR19	PCLKCR19
PCLKCR20	PCLKCR20
PCLKCR21	PCLKCR21
LPMCR	LPMCR
GPIOLPMSEL0	GPIOLPMSEL0
GPIOLPMSEL1	GPIOLPMSEL1
TMR2CLKCTL	TMR2CLKCTL
RESCCLR	RESCCLR
RESC	RESC

Table 49 PERIPH Register Naming Comparison of SYSCTRL Module

G32R501	**Txx320F28004x**
PERIPH_REGS	**PERIPH_AC_REGS**
ADCA_AC	ADCA_AC
ADCB_AC	ADCB_AC
ADCC_AC	ADCC_AC
COMP1AC	CMPSS1_AC
COMP2AC	CMPSS2_AC
COMP3AC	CMPSS3_AC
COMP4AC	CMPSS4_AC
COMP5AC	CMPSS5_AC
COMP6AC	CMPSS6_AC
COMP7AC	CMPSS7_AC
DACA_AC	DACA_AC
DACB_AC	DACB_AC
/	PGA1_AC
/	PGA2_AC

G32R501	Txx320F28004x
PERIPH_REGS	PERIPH_AC_REGS
/	PGA3_AC
/	PGA4_AC
/	PGA5_AC
/	PGA6_AC
/	PGA7_AC
PWM1_AC	EPWM1_AC
PWM2_AC	EPWM2_AC
PWM3_AC	EPWM3_AC
PWM4_AC	EPWM4_AC
PWM5_AC	EPWM5_AC
PWM6_AC	EPWM6_AC
PWM7_AC	EPWM7_AC
PWM8_AC	EPWM8_AC
QEP1_AC	EQEP1_AC
QEP2_AC	EQEP2_AC
CAP1_AC	ECAP1_AC
CAP2_AC	ECAP2_AC
CAP3_AC	ECAP3_AC
CAP4_AC	ECAP4_AC
CAP5_AC	ECAP5_AC
CAP6_AC	ECAP6_AC
CAP7_AC	ECAP7_AC
SDF1_AC	SDFM1_AC
FLB1_AC	CLB1_AC
FLB2_AC	CLB2_AC
FLB3_AC	CLB3_AC
FLB4_AC	CLB4_AC
/	CLA1PROMCRC_AC
SPIA_AC	SPIA_AC
SPIB_AC	SPIB_AC
PMBUS_A_AC	PMBUS_A_AC

G32R501	Txx320F28004x
PERIPH_REGS	**PERIPH_AC_REGS**
LIN_A_AC	LIN_A_AC
CANA_AC	DCANA_AC
CANB_AC	DCANB_AC
/	FSIATX_AC
/	FSIARX_AC
HRPWM_A_AC	HRPWM_A_AC
PERIPH_AC_LOCK	PERIPH_AC_LOCK

Table 50 SPEC Register Naming Comparison of SYSCTRL Module

G32R501	Txx320F28004x
SPEC_REGS	/
EMUBOOTPINCFG	/
EMUBOOTDEFL	/
EMUBOOTDEFH	/
BOOTADDR1	/
BOOTCTRL	/
CFGSMSSEL	/
APBCLKDIVCFG	/
VOS	/

4.1.32. XBAR module

The XBAR module of G32R501 is compatible with the XBAR module of Txx320F28004x. In terms of register naming, the two are the same.

4.1.33. XINT module

G32R501 does not have this module.

4.2. Firmware Library Porting

This section will provide a detailed introduction on how to port the firmware library of Txx320F28004x to G32R501, and the key matters needing to be noticed in this process. We will discuss in detail the differences in firmware library.

The differences are compared in detail in the following aspects:

- Differences in API naming/API quantity/transfer parameter type/number of transfer parameters in driver library function

- Differences in macro definition naming
- Differences in naming of enumeration/enumeration variables

4.2.1. ADC module

The driver library in ADC module of G32R501 is compatible with the driver library in ADC module of Txx320F28004x. But there are differences in the naming of some macro definitions and enumeration variables.

- Naming difference in driver library API
  - No difference
- Differences in driver library API
  - There is no difference in the quantity
  - There is no difference in transfer parameter type
  - There is no difference in the number of transfer parameters
- Naming difference in driver library macro definition
  - Please refer to Table for differencesTable 51
- Differences in driver library enumeration
  - Please refer to Table for differencesTable 52

Table 51 Macro Definition Naming Comparison for ADC Module Driver Library

G32R501	Txx320F28004x
GEEHY_OTP_DEV_KEY	TI_OTP_DEV_KEY
GEEHY_OTP_DEV_PRG_KEY	TI_OTP_DEV_PRG_KEY

Table 52 Differences in Enumeration of ADC Module Driver Library

G32R501	Txx320F28004x
ADC_Trigger enumeration variable	ADC_Trigger enumeration variable

4.2.2. ASYSCTL module

The driver library in ASYSCTL module of G32R501 is compatible with the driver library in ASYSCTL module of Txx320F28004x. But there are differences in API naming, quantity of API, macro definition naming, and enumeration variable naming.

- Naming difference in driver library API

- Please refer to Table for differencesTable 53

- Differences in driver library API
  - Please refer to Table for the difference in the quantity Table 54
  - There is no difference in transfer parameter type
  - There is no difference in the number of transfer parameters
- Naming difference in driver library macro definition
  - Please refer to Table for differencesTable 53
- Differences in driver library enumeration
  - No difference

Table 53 Differences in Naming Prefix of ASYSCTL Module Driver Library

G32R501	Txx320F28004x
Naming prefix	**Naming prefix**
AS_xxxxx	ASYSCTL_xxxxx

Table 54 Differences in the Quantity of API of ASYSCTL Module Driver Library

G32R501	Txx320F28004x
API function name	**API function name**
/	ASysCtl_enableDCDC
/	ASysCtl_disableDCDC
/	ASysCtl_getInductorFaultStatus
/	ASysCtl_getSwitchSequenceStatus
/	ASysCtl_lockDCDC

4.2.3. CAN module

The driver library in CAN module of G32R501 is compatible with the driver library in CAN module of Txx320F28004x. But there are differences in API transfer parameter type and macro definition naming.

- Naming difference in driver library API
  - No difference
- Differences in driver library API
  - There is no difference in the quantity

- Please refer to Table for transfer parameter typeTable 55
- There is no difference in the number of transfer parameters

- Naming difference in driver library macro definition
  - Please refer to Table for differencesTable 56
- Differences in driver library enumeration
  - No difference

Table 55 Differences in Transfer Parameter Type of CAN Module Driver Library API

G32R501	Txx320F28004x
API transfer parameter type	**API transfer parameter type**
CAN_sendMessage(uint32_t base, uint32_t objID, uint16_t msgLen, const uint8_t *msgData)	CAN_sendMessage(uint32_t base, uint32_t objID, uint16_t msgLen, const uint16_t *msgData)
CAN_sendMessage_updateDLC(uint32_t base, uint32_t objID, uint16_t msgLen, const uint8_t *msgData)	CAN_sendMessage_updateDLC(uint32_t base, uint32_t objID, uint16_t msgLen, const uint16_t *msgData)
CAN_readMessage(uint32_t base, uint32_t objID, uint8_t *msgData)	CAN_readMessage(uint32_t base, uint32_t objID, uint16_t *msgData)
bool CAN_readMessageWithID(uint32_t base, uint32_t objID, CAN_MsgFrameType *frameType, uint32_t *msgID, uint8_t *msgData)	bool CAN_readMessageWithID(uint32_t base, uint32_t objID, CAN_MsgFrameType *frameType, uint32_t *msgID, uint16_t *msgData)
CAN_writeDataReg(const uint8_t *const data, uint32_t address, uint32_t size)	CAN_writeDataReg(const uint16_t *const data, uint32_t address, uint32_t size)
CAN_readDataReg(uint8_t *data, const uint32_t address, uint32_t size)	CAN_readDataReg(uint16_t *data, const uint32_t address, uint32_t size)

Table 56 Macro Definition Naming Table for CAN Module Driver Library

G32R501	Txx320F28004x
CAN_STATUS_EWFLG	CAN_STATUS_EWARN
CAN_STATUS_EPFLG	CAN_STATUS_EPASS

4.2.4. CLA module

G32R501 does not have this module.

4.2.5. FLB module/CLB module

The driver library in FLB module of G32R501 is compatible with the driver library in CLB module of Txx320F28004x. But there are differences in API naming, macro definition naming, and enumeration variable naming.

- Naming difference in driver library API
  - Please refer to Table for differencesTable 57
- Differences in driver library API
  - There is no difference in the quantity
  - There is no difference in transfer parameter type
  - There is no difference in the number of transfer parameters
- Naming difference in driver library macro definition
  - Please refer to Table for differencesTable 57
- Differences in driver library enumeration
  - Please refer to Table Table 57 and Table 58 for differences

Table 57 Differences in Naming Prefix of FLB Module Driver Library

G32R501	Txx320F28004x
Naming prefix	**Naming prefix**
FLB_xxxxx	CLB_xxxxx

Table 58 Differences in Enumeration of FLB Module Driver Library

G32R501	Txx320F28004x
Enumeration variable	**Enumeration variable**
FLB_LocalInputMux enumeration variable	CLB_LocalInputMux enumeration variable

FLB_GlobalInputMux enumeration variable	CLB_GlobalInputMux enumeration variable

4.2.6. COMP module/CMPSS module

The driver library in COMP module of G32R501 is compatible with the driver library in CMPSS module of Txx320F28004x. But there are differences in API naming, macro definition naming, and enumeration variable naming.

- Naming difference in driver library API
  - Please refer to Table 59 for differences
- Differences in driver library API
  - There is no difference in the quantity
  - There is no difference in transfer parameter type
  - There is no difference in the number of transfer parameters
- Naming difference in driver library macro definition
  - Please refer to Table 59 for differences
- Differences in driver library enumeration
  - Please refer to Table 59 for differences

Table 59 Differences in Naming Prefix of COMP Module Driver Library

G32R501	**Txx320F28004x**
Naming prefix	**Naming prefix**
COMP_xxxxx	CMPSS_xxxxx

4.2.7. cpu.h

The driver library in cpu.h of G32R501 is compatible with the driver library in cpu.h of Txx320F28004x. But there are differences in macro definition naming.

- Naming difference in driver library API
  - No difference
- Differences in driver library API
  - There is no difference in the quantity
  - There is no difference in transfer parameter type
  - There is no difference in the number of transfer parameters

- Naming difference in driver library macro definition
  - Please refer to Table 60 for differences
- Differences in driver library enumeration
  - No difference

Table 60 Differences in Macro Definition Naming of CPU.H Driver Library

G32R501	Txx320F28004x
WRPRT_DISABLE	EALLOW
WRPRT_ENABLE	EDIS
IDLE_ASM	IDLE

4.2.8. TMR module/CPUTIMER module

The driver library in TMR module of G32R501 is compatible with the driver library in CPUTIMER module of Txx320F28004x. But there are differences in API naming, macro definition naming, and enumeration variable naming.

- Naming difference in driver library API
  - Please refer to Table 61 for differences
- Differences in driver library API
  - There is no difference in the quantity
  - There is no difference in transfer parameter type
  - There is no difference in the number of transfer parameters
- Naming difference in driver library macro definition
  - Please refer to Table 61 for differences
- Differences in driver library enumeration
  - Please refer to Table 61 and Table 62 for differences

Table 61 Differences in Naming Prefix of TMR Module Driver Library

G32R501	Txx320F28004x
Naming prefix	**Naming prefix**
TMR_xxxxx	CPUTIMER_xxxxx
TMR_xxxxx	CPUTimer_xxxxx

Table 62 Differences in Enumeration of TMR Module Driver Library

G32R501	Txx320F28004x
Enumeration variable	**Enumeration variable**
TMR_ClockSource enumeration variable	CPUTimer_ClockSource enumeration variable

4.2.9. DAC module

The driver library in DAC module of G32R501 is compatible with the driver library in DAC module of Txx320F28004x. The driver library has no difference.

4.2.10. DCCOMP module/DCC module

The driver library in DCCOMP module of G32R501 is compatible with the driver library in DCC module of Txx320F28004x. But there are differences in API naming, macro definition naming, and enumeration variable naming.

- Naming difference in driver library API
  - Please refer to Table for differencesTable 63
- Differences in driver library API
  - There is no difference in the quantity
  - There is no difference in transfer parameter type
  - There is no difference in the number of transfer parameters
- Naming difference in driver library macro definition
  - There are prefix differences, which can be referred to in TableTable 63
- Differences in driver library enumeration
  - There are prefix differences, which can be referred to in Table 63

Table 63 Differences in Naming Prefix of DCCOMP Module Driver Library

G32R501	Txx320F28004x
Naming prefix	**Naming prefix**
DCCOMP_xxxxx	DCC_xxxxx

4.2.11. DCS module/DCSM module

The driver library in DCS module of G32R501 is compatible with the driver library in DCSM module of Txx320F28004x. But there are differences in API naming, macro definition naming, and enumeration variable naming.

- Naming difference in driver library API
  - Please refer to Table 64 for differences
- Differences in driver library API
  - There is no difference in the quantity
  - There is no difference in transfer parameter type
  - There is no difference in the number of transfer parameters
- Naming difference in driver library macro definition
  - Please refer to Table 64 and Table 65 for differences
- Differences in driver library enumeration
  - Please refer to Table Table 64 andTable 66 for differences

Table 64 Differences in Naming Prefix of DCS Module Driver Library

G32R501	Txx320F28004x
Naming prefix	**Naming prefix**
DCS_xxxxx	DCSM_xxxxx

Table 65 Differences in Macro Definition Naming of DCS Module Driver Library

G32R501	Txx320F28004x
DCS_LOCKFLG	DCSM_UNSECURE
DCS_READCSPS	DCSM_ARMED

Table 66 Differences in Enumeration of DCS Module Driver Library

G32R501	Txx320F28004x
Enumeration variable	**Enumeration variable**
DCS_SemaphoreZone enumeration variable	DCSM_SemaphoreZone enumeration variable

4.2.12. DMA module

The driver library in DMA module of G32R501 is compatible with the driver library in DMA module of Txx320F28004x. But there are differences in the naming of some enumeration variables.

- Naming difference in driver library API
  - No difference
- Differences in driver library API

- There is no difference in the quantity
- There is no difference in transfer parameter type
- There is no difference in the number of transfer parameters

- Naming difference in driver library macro definition
  - No difference
- Differences in driver library enumeration
  - Please refer to Table 67 for differences

Table 67 Differences in Enumeration of DMA Module Driver Library

G32R501	Txx320F28004x
Enumeration variable	**Enumeration variable**
DMA_Trigger enumeration variable	DMA_Trigger enumeration variable

4.2.13. driver_inclusive_terminology_mapping.h

G32R501 does not have this file.

4.2.14. CAP module/ECAP module

The driver library in CAP module of G32R501 is compatible with the driver library in ECAP module of Txx320F28004x. But there are differences in API naming, macro definition naming, and enumeration variable naming.

- Naming difference in driver library API
  - Please refer to Table 68 for differences
- Differences in driver library API
  - There is no difference in the quantity
  - There is no difference in transfer parameter type
  - There is no difference in the number of transfer parameters
- Naming difference in driver library macro definition
  - Please refer to Table 68 for differences
- Differences in driver library enumeration
  - Please refer to Table 68 and Table 69 for differences

Table 68 Differences in Naming Prefix of CAP Module Driver Library

G32R501	Txx320F28004x
Naming prefix	**Naming prefix**
CAP_xxxxx	ECAP_xxxxx

Table 69 Differences in Enumeration of CAP Module Driver Library

G32R501	Txx320F28004x
Enumeration variable	**Enumeration variable**
CAP_InputCaptureSignals enumeration variable	ECAP_InputCaptureSignals enumeration variable

4.2.15. PWM module/EPWM module

The driver library in PWM module of G32R501 is compatible with the driver library in EPWM module of Txx320F28004x. But there are differences in API naming, macro definition naming, enumeration variable naming, and structure member variable naming.

In the simulation environment, if the user needs to set a breakpoint inside the PWM interrupt function, the user must configure the breakpoint to automatically stop the counter, so that the counter does not continue counting when the simulation is paused. Otherwise, new interrupts may become "blocked," causing the false impression that the interrupt is unresponsive. When initializing the PWM, include the following API function, for example:

DBGMCU_enableDebugHaltSignal(DBGMCU_CORE_CPU0, DBGMCU_DEBUG_HALT_PWM1)

- Naming difference in driver library API
  - Please refer to Table 70 for differences
- Differences in driver library API
  - There is no difference in the quantity
  - There is no difference in transfer parameter type
  - There is no difference in the number of transfer parameters
- Naming difference in driver library macro definition
  - Please refer to Table 70 for differences
- Differences in driver library enumeration
  - Please refer to Table 70 for differences
- Differences in driver library structure
  - Please refer to Table 70 andTable 71 for differences

Table 70 Differences in Naming Prefix of PWM Module Driver Library

G32R501	Txx320F28004x
Naming prefix	**Naming prefix**
PWM_xxxxx	EPWM_xxxxx

Table 71 Differences in the Structure of PWM Module Driver Library

G32R501	Txx320F28004x
Structure member variable	**Structure member variable**
PWM_SignalParams structure member variable apbClkInHz	EPWM_SignalParams structure member variable sysClkInHz

4.2.16. QEP module/EQEP module

The driver library in QEP module of G32R501 is compatible with the driver library in EQEP module of Txx320F28004x. But there are differences in API naming, macro definition naming, and enumeration variable naming.

- Naming difference in driver library API
  - Please refer to Table 72 for differences
- Differences in driver library API
  - There is no difference in the quantity
  - There is no difference in transfer parameter type
  - There is no difference in the number of transfer parameters
- Naming difference in driver library macro definition
  - Please refer to Table 72 for differences
- Differences in driver library enumeration
  - Please refer to Table 72 for differences

Table 72 Differences in Naming Prefix of QEP Module Driver Library

G32R501	Txx320F28004x
Naming prefix	**Naming prefix**
QEP_xxxxx	EQEP_xxxxx

4.2.17. ERAD module

G32R501 does not have this module.

4.2.18. FLASH module

There are significant differences between the driver library of FLASH module of G32R501 and the driver library of FLASH module of Txx320F28004x. Please refer to relevant documents for specific usage. There are differences in the quantity of API and the naming of some enumeration variables.

- Naming difference in driver library API
  - No difference
- Differences in driver library API
  - Please refer to Table 73 for the difference in the quantity
  - There is no difference in transfer parameter type
  - There is no difference in the number of transfer parameters
- Naming difference in driver library macro definition
  - No difference
- Differences in driver library enumeration
  - Please refer to Table for differencesTable 74

Table 73 Differences in the Quantity of API of FLASH Module Driver Library

G32R501	Txx320F28004x
API	**API**
/	Flash_setPumpPowerMode
/	Flash_setPumpActiveGracePeriod
/	Flash_setPumpWakeupTime
/	Flash_isPumpReady
/	Flash_selectLowECCBlock
/	Flash_selectHighECCBlock
Flash_selectLowECCBlock	/
Flash_selectHighECCBlock	/
Flash_flushCache	/
Flash_flushBuffer	/
Flash_getBankMode	/

Table 74 Differences in Enumeration of FLASH Module Driver Library

G32R501	Txx320F28004x
Enumeration variable	**Enumeration variable**
Flash_BankPowerMode enumeration variable	Flash_BankPowerMode enumeration variable
/	Flash_PumpPowerMode enumeration variable
Flash_BankMod enumeration variable	/

4.2.19. GPIO module

The driver library in GPIO module of G32R501 is compatible with the driver library in GPIO module of Txx320F28004x. But there are differences in API naming, quantity of API, and enumeration variable naming.

- Naming difference in driver library API
  - Please refer to Table 75 for differences
- Differences in driver library API
  - Please refer to Table 75 for the difference in the quantity
  - There is no difference in transfer parameter type
  - There is no difference in the number of transfer parameters
- Naming difference in driver library macro definition
  - No difference
- Differences in driver library enumeration
  - Please refer to Table 76 for differences

Table 75 Differences in API Naming/Quantity of GPIO Module Driver Library

G32R501	Txx320F28004x
API	**API**
GPIO_setMasterCore	GPIO_setControllerCore
GPIO_setMaxOutputFreqEnable	/
GPIO_setDrivingCapability	/

Table 76 Differences in Enumeration of GPIO Module Driver Library

G32R501	Txx320F28004x
Enumeration variable	**Enumeration variable**
GPIO_IntType enumeration variable	GPIO_IntType enumeration variable
GPIO_DRIVING_CAPABILITY enumeration variable	/
GPIO_CoreSelect enumeration variable	GPIO_CoreSelect enumeration variable
GPIO_ExternalIntNum enumeration variable	GPIO_ExternalIntNum enumeration variable

4.2.20. HRCAP module

The driver library in HRCAP module of G32R501 is compatible with the driver library in HRCAP module of Txx320F28004x. But there are differences in the quantity of API and the number of transfer parameters.

- Naming difference in driver library API
  - No difference
- Differences in driver library API
  - Please refer to Table 77 for the difference in the quantity
  - There is no difference in transfer parameter type
  - Please refer to Table 77 for the difference in the number of transfer parameters
- Naming difference in driver library macro definition
  - No difference
- Differences in driver library enumeration
  - No difference

Table 77 Differences in Quantity of API and Transfer Parameters of HRCAP Module Driver Library

G32R501	Txx320F28004x
API	**API**
/	HRCAP_configCalibrationPeriod
HRCAP_convertEventTimeStampNanoseconds(uint32_t timeStamp, float32_t scaleFactor, uint32_t APBClock)	HRCAP_convertEventTimeStampNanoseconds(uint32_t timeStamp, float32_t scaleFactor)

4.2.21. HRPWM module

The driver library in HRPWM module of G32R501 is compatible with the driver library in HRPWM module of Txx320F28004x. The driver library has no difference.

4.2.22. I2C module

The driver library in I2C module of G32R501 is compatible with the driver library in I2C module of Txx320F28004x. But there are differences in API naming, macro definition naming, and enumeration variable naming.

- Naming difference in driver library API
  - Please refer to Table 78 for differences
- Differences in driver library API
  - There is no difference in the quantity
  - There is no difference in transfer parameter type
  - There is no difference in the number of transfer parameters
- Naming difference in driver library macro definition
  - Please refer to Table 79 for differences
- Differences in driver library enumeration
  - Please refer to Table Table 80 for differences

Table 78 Differences in API Naming of I2C Module Driver Library

G32R501	Txx320F28004x
API	API
I2C_initMaster	I2C_initController
I2C_setSlaveAddress	I2C_setTargetAddress
I2C_setOwnSlaveAddress	I2C_setOwnAddress

Table 79 Differences in Macro Definition Naming of I2C Module Driver Library

G32R501	Txx320F28004x
I2C_MASTER_SEND_MODE	I2C_CONTROLLER_SEND_MODE
I2C_MASTER_RECEIVE_MODE	I2C_CONTROLLER_RECEIVE_MODE
I2C_SLAVE_SEND_MODE	I2C_TARGET_SEND_MODE
I2C_SLAVE_RECEIVE_MODE	I2C_TARGET_RECEIVE_MODE

I2C_INT_ADDR_SLAVE	I2C_INT_ADDR_TARGET
I2C_STS_INTMASK	I2C_STR_INTMASK
I2C_STS_ADDR_SLAVE	I2C_STS_ADDR_TARGET
I2C_STS_SLAVE_DIR	I2C_STS_TARGET_DIR

Table 80 Differences in Enumeration of I2C Module Driver Library

G32R501	**Txx320F28004x**
I2C_InterruptSource enumeration variable I2C_INTSRC_ADDR_ SLAVE	I2C_InterruptSource enumeration variable I2C_INTSRC_ADDR_TARGET

4.2.23. INTERRUPT module

There are major differences between the driver library of INTERRUPT module of G32R501 and the driver library of INTERRUPT module (PIE) of Txx320F28004x. Please refer to Section 7.2 for specific usage.

4.2.24. LIN module

The driver library in LIN module of G32R501 is compatible with the driver library in LIN module of Txx320F28004x. But there are differences in API naming, macro definition naming, and enumeration variable naming.

- Naming difference in driver library API
  - Please refer to Table 81 for differences
- Differences in driver library API
  - There is no difference in the quantity
  - There is no difference in transfer parameter type
  - There is no difference in the number of transfer parameters
- Naming difference in driver library macro definition
  - Please refer to Table 82 for differences
- Differences in driver library enumeration
  - Please refer to Table 83 for differences

Table 81 Differences in API Naming of LIN Module Driver Library

G32R501	**Txx320F28004x**
API	**API**

LIN_setIDSlaveTask	LIN_setIDResponderTask
xxxUARTxxx	xxxSCIxxx

Table 82 Differences in Macro Definition Naming of LIN Module Driver Library

G32R501	Txx320F28004x
LIN_UART_ALL_ERRORS	LIN_SCI_ALL_ERRORS
LIN_UART_FRAME_ERROR	LIN_SCI_FRAME_ERROR
LIN_UART_PARITY_ERROR	LIN_SCI_PARITY_ERROR
LIN_UART_BREAK_ERROR	LIN_SCI_BREAK_ERROR
LIN_UART_INT_BREAK	LIN_SCI_INT_BREAK
LIN_UART_INT_WAKEUP	LIN_SCI_INT_WAKEUP
LIN_UART_INT_TX	LIN_SCI_INT_TX
LIN_UART_INT_RX	LIN_SCI_INT_RX
LIN_UART_INT_TX_DMA	LIN_SCI_INT_TX_DMA
LIN_UART_INT_RX_DMA	LIN_SCI_INT_RX_DMA
LIN_UART_INT_RX_DMA_ALL	LIN_SCI_INT_RX_DMA_ALL
LIN_UART_INT_PARITY	LIN_SCI_INT_PARITY
LIN_UART_INT_OVERRUN	LIN_SCI_INT_OVERRUN
LIN_UART_INT_FRAME	LIN_SCI_INT_FRAME
LIN_UART_INT_ALL	LIN_SCI_INT_ALL

Table 83 Differences in Enumeration of LIN Module Driver Library

G32R501	Txx320F28004x
LIN_UARTCommMode enumeration variable	LIN_SCICommMode enumeration variable
LIN_LINMode enumeration variable	LIN_LINMode enumeration variable
LIN_MessageFilter enumeration variable	LIN_MessageFilter enumeration variable
LIN_UARTParityType enumeration variable	LIN_SCIParityType enumeration variable
LIN_UARTStopBits enumeration variable	LIN_SCIStopBits enumeration variable

4.2.25. MEMCFG module

G32R501 does not have this module.

4.2.26. PGA module

G32R501 does not have this module.

4.2.27. pin_map_legacy.h / pin_map.h

Due to the differences in pin compatibility between G32R501 and Txx320F28004x, there are differences between the driver file pin_map_legacy.h / pin_map.h of G32R501 and the driver file pin_map_legacy.h / pin_map.h of Txx320F28004x. The content of these two files is the definition of reuse macro of GPIO, and there are many differences. Please refer to the files for details.

Note: For pin compatibility, please refer to Chapter 2, Section 2 of *G325Rx1 and Txx320F28004x Difference Manual*.

4.2.28. PMBUS module

The driver library in PMBUS module of G32R501 is compatible with the driver library in PMBUS module of Txx320F28004x. But there are differences in API naming, macro definition naming, and enumeration variable naming.

- Naming difference in driver library API
  - Please refer to Table 84 for differences
- Differences in driver library API
  - There is no difference in the quantity
  - There is no difference in transfer parameter type
  - There is no difference in the number of transfer parameters
- Naming difference in driver library macro definition
  - Please refer to Table 85 for differences
- Differences in driver library enumeration
  - Please refer to Table 86 for differences

Table 84 Differences in API Naming of PMBUS Module Driver Library

G32R501	Txx320F28004x
API	API
PMBus_initSlaveMode	PMBus_initTargetMode
PMBus_configSlave	PMBus_configTarget
PMBus_putSlaveData	PMBus_putTargetData
PMBus_initMasterMode	PMBus_initControllerMode
PMBus_putMasterData	PMBus_putControllerData

PMBus_configMaster	PMBus_configController
PMBus_setSlaveAddress	PMBus_setTargetAddress

Table 85 Differences in Macro Definition Naming of PMBUS Module Driver Library

G32R501	Txx320F28004x
PMBUS_MASTER_ENABLE_TXPROMEN	PMBUS_CONTROLLER_ENABLE_PRC_CALL
PMBUS_MASTER_ENABLE_GCOMEN	PMBUS_CONTROLLER_ENABLE_GRP_CMD
PMBUS_MASTER_ENABLE_PECBYTEEN	PMBUS_CONTROLLER_ENABLE_PEC
PMBUS_MASTER_ENABLE_BYTECFG	PMBUS_CONTROLLER_ENABLE_EXT_CMD
PMBUS_MASTER_ENABLE_COMCEN	PMBUS_CONTROLLER_ENABLE_CMD
PMBUS_MASTER_ENABLE_READ	PMBUS_CONTROLLER_ENABLE_READ
PMBUS_INT_BUS_IDLE	PMBUS_INT_BUS_FREE
PMBUS_INT_SLAVE_ADDR_READY	PMBUS_INT_TARGET_ADDR_READY
PMBUS_INTSRC_AVAFLG	PMBUS_INTSRC_BUS_FREE
PMBUS_INTSRC_CLKLTOFLG	PMBUS_INTSRC_CLK_LOW_TIMEOUT
PMBUS_INTSRC_READDATA	PMBUS_INTSRC_DATA_READY
PMBUS_INTSRC_ADITDATA	PMBUS_INTSRC_DATA_REQUEST
PMBUS_INTSRC_RDYRSADDR	PMBUS_INTSRC_TARGET_ADDR_READY
PMBUS_INTSRC_ENDFLG	PMBUS_INTSRC_EOM
PMBUS_INTSRC_ALERTTRAN	PMBUS_INTSRC_ALERT
PMBUS_INTSRC_CTRLTRAN	PMBUS_INTSRC_CONTROL
PMBUS_INTSRC_LOSTCTRL	PMBUS_INTSRC_LOST_ARB
PMBUS_INTSRC_CLKHFLG	PMBUS_INTSRC_CLK_HIGH_DETECT
PMBUS_SLAVE_ENABLE_SADDRACKEN	PMBUS_TARGET_ENABLE_MANUAL_ACK
PMBUS_SLAVE_ENABLE_PEC_PROCESSING	PMBUS_TARGET_ENABLE_PEC_PROCESSING
PMBUS_SLAVE_TXPEC	PMBUS_TARGET_TRANSMIT_PEC
PMBUS_SLAVE_ENABLE_COMC	PMBUS_TARGET_ENABLE_MANUAL_CMD_ACK
PMBUS_SLAVE_DISABLE_ADDRESS_MASK	PMBUS_TARGET_DISABLE_ADDRESS_MASK
PMBUS_SLAVE_AUTOACK_1_BYTES	PMBUS_TARGET_AUTO_ACK_1_BYTES
PMBUS_SLAVE_AUTOACK_2_BYTES	PMBUS_TARGET_AUTO_ACK_2_BYTES
PMBUS_SLAVE_AUTOACK_3_BYTES	PMBUS_TARGET_AUTO_ACK_3_BYTES
PMBUS_SLAVE_AUTOACK_4_BYTES	PMBUS_TARGET_AUTO_ACK_4_BYTES
PMBUS_STSMRSTEN	PMBUS_RESET
PMBUS_ENABLE_SLAVE_ALERTLOW	PMBUS_ENABLE_TARGET_ALERT
PMBUS_SET_CLKLTOCFG	PMBUS_SET_BUS_LO_INT_FALLING_EDGE
PMBUS_CTRLCFG	PMBUS_SET_CNTL_INT_RISING_EDGE
PMBUS_ENABLE_THRUENA	PMBUS_ENABLE_IBIAS_A_EN

G32R501	Txx320F28004x
PMBUS_ENABLE_THRUENB	PMBUS_ENABLE_IBIAS_B_EN
PMBUS_DISABLE_DISCLKLTO	PMBUS_DISABLE_CLK_LOW_TIMEOUT
PMBUS_ENABLE_SLAVE_MODE	PMBUS_ENABLE_TARGET_MODE
PMBUS_ENABLE_MASTER_MODE	PMBUS_ENABLE_CONTROLLER_MODE
PMBUS_APB_FREQ_MIN	PMBUS_SYS_FREQ_MIN
PMBUS_APB_FREQ_MAX	PMBUS_SYS_FREQ_MAX

Table 86 Differences in Enumeration of PMBUS Module Driver Library

G32R501	Txx320F28004x
PMBUS_IMASKEdge enumeration variable	PMBus_intEdge enumeration variable

4.2.29. UART module/SCI module

The driver library in UART module of G32R501 is compatible with the driver library in SCI module of Txx320F28004x. But there are differences in API naming, API transfer parameter type, macro definition naming, and enumeration variable naming.

During the UART initialization process, if the RX-related IO pin is floating, invalid data may be received after initialization. This occurs because the GPIO is not configured with pull-up or pull-down resistors, causing the signal level from the IO pin to the internal digital circuit to be unpredictable (due to analog electrical characteristics). Therefore, configuring the RX GPIO pin with a pull-up resistor during initialization can prevent this issue.

- Naming difference in driver library API
  - Please refer to Table 87 for differences
- Differences in driver library API
  - There is no difference in the quantity
  - Please refer to Table 88 for transfer parameter type
  - There is no difference in the number of transfer parameters
- Naming difference in driver library macro definition
  - Please refer to Table 87 for differences
- Differences in driver library enumeration
  - Please refer to Table 87 for differences

Table 87 Differences in Naming Prefix of UART Module Driver Library

G32R501	Txx320F28004x

Naming prefix	Naming prefix
UART_xxxxx	SCI_xxxxx

Table 88 Differences in API Transfer Parameter Types of UART Module Driver Library

G32R501	Txx320F28004x
API	**API**
UART_writeCharBlockingFIFO(uint32_t base, uint8_t data)	SCI_writeCharBlockingFIFO(uint32_t base, uint16_t data)
UART_writeCharBlockingNonFIFO(uint32_t base, uint8_t data)	SCI_writeCharBlockingNonFIFO(uint32_t base, uint16_t data)
UART_writeCharNonBlocking(uint32_t base, uint8_t data)	SCI_writeCharNonBlocking(uint32_t base, uint16_t data)
static inline uint8_t UART_readCharNonBlocking(uint32_t base)	static inline uint16_t SCI_readCharNonBlocking(uint32_t base)
UART_writeCharArray(uint32_t base, const uint8_t * const array, uint16_t length)	SCI_writeCharArray(uint32_t base, const uint16_t * const array, uint16_t length)
UART_readCharArray(uint32_t base, uint8_t * const array, uint16_t length)	SCI_readCharArray(uint32_t base, uint16_t * const array, uint16_t length)

Table 89 Differences in Enumeration of UART Module Driver Library

G32R501	Txx320F28004x
Enumeration/Enumeration variable	**Enumeration/Enumeration variable**
UART_xxxxx	SCI_xxxxx

4.2.30. SDF module/SDFM module

The driver library in SDF module of G32R501 is compatible with the driver library in SDFM module of Txx320F28004x. But there are differences in API naming, macro definition naming, and enumeration variable naming.

- Naming difference in driver library API
  - Please refer to Table 90 and Table 91 for differences
- Differences in driver library API

- There is no difference in the quantity
- There is no difference in transfer parameter type
- There is no difference in the number of transfer parameters

- Naming difference in driver library macro definition
  - Please refer to Table 90 and Table 92 for differences
- Differences in driver library enumeration
  - Please refer to Table 90 for differences

Table 90 Differences in Naming Prefix of SDF Module Driver Library

G32R501	Txx320F28004x
Naming prefix	**Naming prefix**
SDF_xxxxx	SDFM_xxxxx

Table 91 Differences in API Naming of SDF Module Driver Library

G32R501	Txx320F28004x
SDF_enableMasterInterrupt	SDFM_enableMainInterrupt
SDF_disableMasterInterrupt	SDFM_disableMainInterrupt
SDF_enableMasterFilter	SDFM_enableMainFilter
SDF_disableMasterFilter	SDFM_disableMainFilter

Table 92 Differences in Macro Definition Naming of SDF Module Driver Library

G32R501	Txx320F28004x
SDF_MASTER_INTERRUPT_FLAG	SDFM_MAIN_INTERRUPT_FLAG

4.2.31. SPI module

The driver library in SPI module of G32R501 is compatible with the driver library in SPI module of Txx320F28004x. But there are differences in API naming, and enumeration variable naming.

- Naming difference in driver library API
  - Please refer to Table 93 for differences
- Differences in driver library API
  - There is no difference in the quantity
  - There is no difference in transfer parameter type
  - There is no difference in the number of transfer parameters

- Naming difference in driver library macro definition
  - No difference
- Differences in driver library enumeration
  - Please refer to Table 94 for differences

Table 93 Differences in API Naming of SPI Module Driver Library

G32R501	Txx320F28004x
API	API
SPI_setSTESignalPolarity	SPI_setPTESignalPolarity

Table 94 Differences in Enumeration of SPI Module Driver Library

G32R501	Txx320F28004x
Enumeration variable	Enumeration variable
SPI_Mode enumeration variable	SPI_Mode enumeration variable
SPI_STEPolarity enumeration variable	SPI_PTEPolarity enumeration variable

4.2.32. SYSCTL module

The driver library in SYSCTL module of G32R501 is compatible with the driver library in SYSCTL module of Txx320F28004x. But there are differences in API naming, API returned value, the quantity of API, macro definition naming, and enumeration variable naming.

- Naming difference in driver library API
  - Please refer to Table 95 for differences
- Differences in driver library API
  - For the differences in returned value, please refer to Table 95
  - Please refer to Table 95 for the difference in the quantity
  - There is no difference in transfer parameter type
  - There is no difference in the number of transfer parameters
- Naming difference in driver library macro definition
  - Please refer to Table 96 for difference
- Differences in driver library enumeration
  - Please refer to Table 97 for differences

Table 95 Differences in API of SYSCTL Module Driver Library

G32R501	Txx320F28004x
API	**API**
SysCtl_delay	SYSCTL_DELAY (macro)
static void SysCtl_pollX1Counter(void)	static bool SysCtl_pollX1Counter(void)
SysCtl_getAPBClock	/
SysCtl_setAPBClk	/
SysCtl_enableVOS	/
SysCtl_disableVOS	/
SysCtl_setLDOVoltage	/
SysCtl_enableBootRomPrefetch	/
SysCtl_disableBootRomPrefetch	/
SysCtl_enableSecureRomPrefetch	/
SysCtl_disableSecureRomPrefetch	/
SysCtl_setBootRomLatency	/
SysCtl_setSecureRomLatency	/
/	SysCtl_isWatchdogEnabled
/	SysCtl_getEfuseError
/	SysCtl_getPIEVErrAddr
SysCtl_setCPU1BootAddress	/
SysCtl_enableBootCPU1	/

Table 96 Differences in Macro Definition of SYSCTL Module Driver Library

G32R501	Txx320F28004x
SYSCTL_WDT_CHKBITS	SYSCTL_WD_CHKBITS
SYSCTL_WDT_ENRSTKEY	SYSCTL_WD_ENRSTKEY
SYSCTL_WDT_RSTKEY	SYSCTL_WD_RSTKEY
/	SYSCTL_DEVICECAL_CONTEXT_SAVE
/	SYSCTL_DEVICECAL_CONTEXT_RESTORE
/	SYSCTL_NMI_RAMUNCERR
/	SYSCTL_NMI_PIEVECTERR
SYSCTL_NMI_FLBNMI	SYSCTL_NMI_CLBNMI

Table 97 Differences in Enumeration of SYSCTL Module Driver Library

G32R501	Txx320F28004x
Enumeration variable	**Enumeration variable**
SysCtl_PeripheralPCLOCKCR enumeration variable	SysCtl_PeripheralPCLOCKCR enumeration variable
SysCtl_PeripheralSWRST enumeration variable	SysCtl_PeripheralSOFTPRES enumeration variable
SysCtl_WDPredivider enumeration variable	SysCtl_WDPredivider enumeration variable
SysCtl_WDPrescaler enumeration variable	SysCtl_WDPrescaler enumeration variable
SysCtl_WDMode enumeration variable	SysCtl_WDMode enumeration variable
SysCtl_APBCLKPrescaler enumeration variable	/
SysCtl_AccessPeripheral enumeration variable	SysCtl_AccessPeripheral enumeration variable
SysCtl_AccessMaster enumeration variable	SysCtl_AccessController enumeration variable
SysCtl_ClockOut enumeration variable	SysCtl_ClockOut enumeration variable
SysCtl_SyncInput enumeration variable	SysCtl_SyncInput enumeration variable
SysCtl_SyncInputSource enumeration variable	SysCtl_SyncInputSource enumeration variable
SysCtl_SyncOutputSource enumeration variable	SysCtl_SyncOutputSource enumeration variable
SysCtl_Timer2ClkDivider enumeration name	SysCtl_Cputimer2ClkDivider enumeration name
SysCtl_Timer2ClkSource enumeration variable	SysCtl_Cputimer2ClkSource enumeration variable
SysCtl_LDOVolScaler enumeration variable	/

4.2.33. VERSION module

The driver library in VERSION module of G32R501 is compatible with the driver library in VERSION module of Txx320F28004x. The driver library has no difference.

4.2.34. XBAR module

The driver library in XBAR module of G32R501 is compatible with the driver library in XBAR module of Txx320F28004x. But there are differences in API naming, macro definition naming, and enumeration variable naming.

- Naming difference in driver library API
  - Please refer to Table 98 for differences
- Differences in driver library API
  - There is no difference in the quantity
  - There is no difference in transfer parameter type
  - There is no difference in the number of transfer parameters

- Naming difference in driver library macro definition
  - Please refer to Table 99 for differences
- Differences in driver library enumeration
  - Please refer toTable 100 for differences

Table 98 Differences in API Naming of XBAR Module Driver Library

G32R501	Txx320F28004x
API	**API**
XBAR_setPWMMuxConfig	XBAR_setEPWMMuxConfig
XBAR_setFLBMuxConfig	XBAR_setCLBMuxConfig
XBAR_enablePWMMux	XBAR_enableEPWMMux
XBAR_disablePWMMux	XBAR_disableEPWMMux
XBAR_invertPWMSignal	XBAR_invertEPWMSignal
XBAR_lockPWM	XBAR_lockEPWM
XBAR_enableFLBMux	XBAR_enableCLBMux
XBAR_disableFLBMux	XBAR_disableCLBMux
XBAR_invertFLBSignal	XBAR_invertCLBSignal

Table 99 Differences in Macro Definition Naming of XBAR Module Driver Library

G32R501	Txx320F28004x
XBAR_PWM_CFG_REG_BASE	XBAR_EPWM_CFG_REG_BASE
XBAR_PWM_EN_REG_BASE	XBAR_EPWM_EN_REG_BASE
XBAR_FLB_CFG_REG_BASE	XBAR_CLB_CFG_REG_BASE
XBAR_FLB_EN_REG_BASE	XBAR_CLB_EN_REG_BASE

Table 100 Differences in Enumeration Body of XBAR Module Driver Library

G32R501	Txx320F28004x
Enumeration variable	**Enumeration variable**
XBAR_OutputMuxConfig enumeration variable	XBAR_OutputMuxConfig enumeration variable
XBAR_PWMMuxConfig enumeration variable	AR_EPWMMuxConfig enumeration variable
XBAR_FLBMuxConfig enumeration variable	XBAR_CLBMuxConfig enumeration variable
XBAR_InputFlag enumeration variable	XBAR_InputFlag enumeration variable

4.3. Common Problems and Solutions

4.3.1. Differences in bit field naming

Due to the inconsistent naming of registers in some compatible modules, the example 1 below is a sample of the register of G32R501 UART module compatible with the register of Txx320F28004x SCI module. The function implemented by them is consistent, and they can be directly replaced in practical application code.

Example 1:

The following code for G32R501:

```
UartaRegs.UARTCCR.all = 0x0007;
UartaRegs.UARTCTL1.all = 0x0003;
UartaRegs.UARTCTL2.all = 0x0003;
UartaRegs.UARTCTL2.bit.TXINTENA = 1;
UartaRegs.UARTCTL2.bit.RXBKINTENA = 1;
```

Equivalent to the code in Txx320F28004x:

```
SciaRegs.SCICCR.all = 0x0007;
SciaRegs.SCICTL1.all = 0x0003;
SciaRegs.SCICTL2.all = 0x0003;
SciaRegs.SCICTL2.bit.TXINTENA = 1;
SciaRegs.SCICTL2.bit.RXBKINTENA = 1;
```

The solution is to modify the corresponding register naming of the compatible module, and corresponding bit field naming.

4.3.2. Naming difference in firmware library API

Due to the inconsistent API naming of the firmware library of some compatible modules, the example 1 below is an example of the firmware library API of G32R501 TMR module compatible with the firmware library API of Txx320F28004x CPUTIMER module. The function implemented by them is consistent, and they can be directly replaced in practical application code.

Example 1:

The following code for G32R501:

```
TMR_startTimer (TMR0_BASE);
TMR_startTimer (TMR1_BASE);
TMR_startTimer (TMR2_BASE);
```

Equivalent to the code in Txx320F28004x:

```
CPUTimer_startTimer (CPUTIMER0_BASE);
CPUTimer_startTimer (CPUTIMER1_BASE);
CPUTimer_startTimer (CPUTIMER2_BASE);
```

The solution is to modify the naming of the firmware library API of the compatible module.

4.3.3. Differences in transfer parameter type of firmware library API

Due to the inconsistent transfer parameter type of the firmware library API of some compatible modules, the example 1 below is an example of the firmware library API of G32R501 CAN module compatible with the firmware library of Txx320F28004x CAN module. Example 2 is an example of the firmware library API of G32R501 UART module compatible with the firmware library API of Txx320F28004x SCI module. The function implemented by them is consistent, and they can be directly replaced in practical application code.

Example 1:

The following code for G32R501:

```
extern void CAN_sendMessage(uint32_t base, uint32_t objID, uint16_t msgLen,
const uint8_t *msgData);

uint8_t txMsgData[2];
//
// Send CAN message data from
// message object 1
//
CAN_sendMessage(myCAN0_BASE, 1, MSG_DATA_LENGTH, txMsgData);
```

Equivalent to the code in Txx320F28004x:

```
extern void CAN_sendMessage(uint32_t base, uint32_t objID, uint16_t msgLen,
const uint16_t *msgData);

uint16_t txMsgData[2];
//
// Send CAN message data from
// message object 1
//
CAN_sendMessage(myCAN0_BASE, 1, MSG_DATA_LENGTH, txMsgData);
```

The solution is to modify the declaration type of the actual parameter so that when calling the function, it conforms to the API transfer parameter type of firmware library of the compatible module.

Example 2:

The following code for G32R501:

```
extern void UART_writeCharArray(uint32_t base, const uint8_t * const array,
uint16_t length);

//
// Send starting message.
//
msg = "\r\n\n\nHello World!\0";
UART_writeCharArray(UARTA_BASE, (uint8_t *)msg, 17);
```

Equivalent to the code in Txx320F28004x:

```
extern void SCI_writeCharArray(uint32_t base, const uint16_t * const array,
uint16_t length);

//
// Send starting message.
//
msg = "\r\n\n\nHello World!\0";
SCI_writeCharArray(SCIA_BASE, (uint16_t*)msg, 17);
```

The solution is to modify the transfer parameter type of API of firmware library of the compatible module when calling the function.

4.3.4. Differences in the quantity of transfer parameters of firmware library API

Due to the inconsistent quantity of transfer parameters of firmware library API of some compatible modules, the example 1 below is an example of the firmware library API of G32R501 HRCAP module compatible with the firmware library API of Txx320F28004x HRCAP module. The function implemented by them is consistent, and they can be directly replaced in practical application code.

Example 1:

The following code for G32R501:

```
static inline float32_t HRCAP_convertEventTimeStampNanoseconds(
uint32_t timeStamp, float32_t scaleFactor, uint32_t APBClock)

//
// Convert counts to nanoseconds
// using the scale factor
```

```
//
onTime1 = HRCAP_convertEventTimeStampNanoseconds(
absCountOn1, hrcapCalResult.scaleFactor, APBClock);
```

Equivalent to the code in Txx320F28004x:

```
static inline float32_t HRCAP_convertEventTimeStampNanoseconds(
uint32_t timeStamp, float32_t scaleFactor)

//
// Convert counts to nanoseconds
// using the scale factor
//
onTime1 = HRCAP_convertEventTimeStampNanoseconds(
absCountOn1, hrcapCalResult.scaleFactor);
```

The solution is to add the quantity of transfer parameters of API of firmware library of the compatible module when calling the function.

4.3.5. Differences in macro definition

Due to the inconsistent macro definition naming of the firmware library of some compatible modules, the example 1 below is an example of the macro definition of G32R501 firmware library compatible with the macro definition of the firmware library of Txx320F28004x. The function implemented by them is consistent, and they can be directly replaced in practical application code.

Example 1:

The following code for G32R501:

```
//
// Define to disable writes to protected registers
//
#ifndef WRPRT_DISABLE
#define WRPRT_DISABLE   __wrprt_disable()

#endif

//
// Define to allow writes to protected registers
//
#ifndef WRPRT_ENABLE
#define WRPRT_ENABLE __wrprt_enable()
#endif
```

```
//
// code
//
WRPRT_DISABLE;
...
WRPRT_ENABLE;
```

Equivalent to the code in Txx320F28004x:

```
//
// Define to allow writes to protected registers
//
#ifndef EALLOW
#ifndef __TMS320C28XX_CLA__
#define EALLOW __eallow()
#else
#define EALLOW __meallow()
#endif
#endif

//
// Define to disable writes to protected registers
//
#ifndef EDIS
#ifndef __TMS320C28XX_CLA__
#define EDIS    __edis()
#else
#define EDIS    __medis()
#endif
#endif

//
// code
//
EALLOW;
...
EDIS;
```

The solution is to modify the macro definition naming of the firmware library of the compatible *module.*

4.3.6. Differences in enumeration variables of firmware library

Due to the inconsistent enumeration variable naming of firmware library of some compatible modules, the example 1 below is an example of the enumeration variable of firmware library of G32R501 ADC module compatible with the enumeration variable of firmware library of Txx320F28004x ADC module. The function implemented by them is consistent, and they can be directly replaced in practical application code.

Example 1:

The following code for G32R501:

```
typedef enum
{
    ...
    ADC_TRIGGER_PWM1_SOCA   = 5U,
    ADC_TRIGGER_PWM1_SOCB   = 6U,
    ADC_TRIGGER_PWM2_SOCA   = 7U,
    ADC_TRIGGER_PWM2_SOCB   = 8U,
    ...
} ADC_Trigger;

//
// code
//
ADC_setupSOC(myADC0_BASE, ADC_SOC_NUMBER0, ADC_TRIGGER_PWM1_SOCA,
ADC_CH_ADCIN0, 8U);
```

Equivalent to the code in Txx320F28004x:

```
typedef enum
{
    ...
    ADC_TRIGGER_EPWM1_SOCA   = 5U,
    ADC_TRIGGER_EPWM1_SOCB   = 6U,
    ADC_TRIGGER_EPWM2_SOCA   = 7U,
    ADC_TRIGGER_EPWM2_SOCB   = 8U,
    ...
} ADC_Trigger;
```

```
//
// code
//
ADC_setupSOC(myADC0_BASE, ADC_SOC_NUMBER0, ADC_TRIGGER_EPWM1_SOCA,
ADC_CH_ADCIN0, 8U);
```

The solution is to modify the enumeration variable naming of firmware library of the compatible module when calling the function.

5. Calibration Library Porting

5.1. SFO Library Software Porting

Similar to Txx320F28004x, G32R5xx provides a C callable library SFO library (SFO.lib) for calibrating PWM, which includes an SFO function. This function utilizes hardware and determines the optimal HRP scaling factor. During HRPWM operation, this SFO function helps dynamically determine the HRP step for each PWMCLK cycle.

This program drives the HRP calibration module to run SFO diagnostics and determines the appropriate HRP scale factor for the device at any given time, i.e., the HRP step per PWMCLK coarse step. Assuming PWMCLK = TBCLK = 100 MHz and the HRP step is 150 ps, the typical scale factor value at 100 MHz is 66 HRP steps per 10 ns (TBCLK unit).

SFO() can be called at any time on the HRP calibration module to run the SFO diagnostics.

If the PWM channel is operating in HRPWM mode, the function can be called in the background at any time. Its function is to utilize the diagnostic logic in the HRP calibration module, and the resulting scale factor can then be applied to all PWM channels running in HRPWM mode.

This function returns values to indicate different situations:

- The program returns 00: Indicates that calibration is still running
- The program returns 01: Indicates that calibration is completed and a new scale factor has been calculated
- The program returns 10: Indicates an error, and the HRP scale factor is greater than each coarse step

Usage of SFO.lib:

- Add SFO.lib and “Include” files to the project
- Variable declaration:

For example: int MEP_ScaleFactor; // Global variables used by SFO library

- MEP_ScaleFactor initialization:

The SFO() function shall be called to calculate the initial MEP_ScaleFactor value.

- Application code:

When the application is running, the SFO function shall periodically rerun as part of a slower background loop.

6. Math Library Porting

In embedded systems, the use of the math library is crucial for implementing complex mathematical operations and algorithms. When migrating from the Txx320F28004x series to the G32R501 series microcontrollers, adaptation and optimization is required for the math library to fully use the hardware characteristics of the target platform.

This chapter will focus on the porting methods of various mathematical libraries. We will emphatically discuss the compatibility of math libraries and how our self-developed Zidian extension unit can be combined with math libraries for computation acceleration. The R501 math library includes: fixed-point arithmetic library (Fixed_Point), floating-point unit (FPU), vector computing unit (VCU), fixed-point arithmetic library (Fix32math, benchmarking IQmath), fast floating-point arithmetic library (FPUfastRTS), and Zidian extension unit.

The general steps for porting a math library are as follows:

- Analyze the existing code:
  - Browse and understand the math library used in existing code, and record the math library functions used.
- Compare the function naming:
  - Refer to the function naming table for each math library given in the following sections, and find the corresponding math library functions.
- Modify the code:
  - Replace the math library functions used in the existing code with the math library functions of G32R501.

In addition, in the porting process, in order to speed up the computation, the data required for program calculation (e.g. FFT buffers) needs to be stored in the specific memory area. The Txx320F28004x SDK places the array in the corresponding memory segment through the cmd file. In R501, through chained files, users can store the corresponding data in the designated memory area according to the following example codes.

```
// Create the FFT buffers, and stored in DCTM
SECTION_DTCM_BSS

int32_t ipcb[2*FFT_SIZE];
```

6.1. Fixed_Point

The fixed-point arithmetic is widely used in embedded systems for control and signal processing tasks. This section will introduce how to migrate the fixed-point arithmetic library from Txx320F28004x to G32R501.

In TI Fixed_Point library, there are four different configurations of ISA_C2800, ISA_C2800_EABI, ISA_C28FPU32 and ISA_C28FPU32_EABI, corresponding to four different lib library files, i.e. c28x_fixedpoint_dsp_library.lib, c28x_fixedpoint_dsp_library_eabi.lib, c28x_fixedpoint_dsp_library_fpu32.lib, and c28x_fixedpoint_dsp_library_fpu32_eabi.lib. In Txx320F28004x, the c28x_fixedpoint_dsp_library_fpu32_eabi.lib is used, and the Fixed_Point library of G32R501 is fully compatible with the Fixed_Pointlibrary used in Txx320F28004x. The following table shows a list of R501 implementation functions and Txx320F28004x implementation functions:

Table 101 List of Implementation Functions of R501 Fixed_Point and Implementation Functions of Competitive Products

G32R501		Txx320F28004x	
Function name	**Function name**	**Function name**	**Function prototype**
Bit Reverse Modules			
CFFT32_brev	void CFFT32_brev(int32_t *src, int32_t *dst, uint16_t size)	CFFT32_brev	void CFFT32_brev(int32 *src, int32 *dst, int16 size);
RFFT32_brev	void RFFT32_brev(int32_t *src, int32_t *dst, uint16_t size)	RFFT32_brev	void RFFT32_brev(int32 *src, int32 *dst, int16 size);
RFFT32_brev_RT	void RFFT32_brev_RT(void *handle)	RFFT32_brev_RT	void RFFT32_brev_RT(void *);
FFT Modules			
FFT32_calc	void FFT32_calc(void *handle)	FFT32_calc	void FFT32_calc(void *);
FFT32_init	void FFT32_init(void *handle)	FFT32_init	void FFT32_init(void *);
FFT32_izero	void FFT32_izero(void *handle)	FFT32_izero	void FFT32_izero(void *);
CFFT32_mag	void CFFT32_mag(void *handle)	CFFT32_mag	void CFFT32_mag(void *);
CFFT32_win	void CFFT32_win(void *handle)	CFFT32_win	void CFFT32_win(void *);
RFFT32_split	void RFFT32_split(void *handle)	RFFT32_split	void RFFT32_split(void *);
RFFT32_mag	void RFFT32_mag(void *handle)	RFFT32_mag	void RFFT32_mag(void *);
RFFT32_win	void RFFT32_win(void *handle)	RFFT32_win	void RFFT32_win(void *);
FIR Modules			
FIR16_init	void FIR16_init(void *handle)	FIR16_init	void FIR16_init(void *);
FIR16_calc	void FIR16_calc(void *handle)	FIR16_calc	void FIR16_calc(void *);
FIR16_alt_init	void FIR16_alt_init(void *handle)	FIR16_Alt_init	void FIR16_Alt_init(void *);
FIR16_alt_calc	void FIR16_alt_calc(void *handle)	FIR16_Alt_calc	void FIR16_Alt_calc(void *);
FIR32_init	void FIR32_init(void *handle)	FIR32_init	void FIR32_init(void *);
FIR32_calc	void FIR32_calc(void *handle)	FIR32_calc	void FIR32_calc(void *);
FIR32_alt_init	void FIR32_alt_init(void *handle)	FIR32_Alt_init	void FIR32_Alt_init(void *);
FIR32_alt_calc	void FIR32_alt_calc(void *handle)	FIR32_Alt_calc	void FIR32_Alt_calc(void *);

G32R501		Txx320F28004x	
Function name	**Function name**	**Function name**	**Function prototype**
IIR Modules			
IIR5BIQ16_init	void IIR5BIQ16_init(IIR5BIQ16 *IIR5BIQ16_Handle)	IIR5BIQ16_init	void IIR5BIQ16_init(void *);
IIR5BIQ16_calc	void IIR5BIQ16_calc(IIR5BIQ16 *IIR5BIQ16_Handle)	IIR5BIQ16_calc	void IIR5BIQ16_calc(void *);
IIR5BIQ32_init	void IIR5BIQ32_init(IIR5BIQ32 *IIR5BIQ32_Handle)	IIR5BIQ32_init	void IIR5BIQ32_init(void *);
IIR5BIQ32_calc	void IIR5BIQ32_calc(IIR5BIQ32 *IIR5BIQ32_Handle)	IIR5BIQ32_calc	void IIR5BIQ32_calc(void *)

The implementation function of the Fixed_Point library is declared and the related structure parameters is defined in the file fft.h. Users can port the program to G32R501 according to the function list.

An example is as follows:

```
// Declare and initialize the structure object.
// Use the CFFT32_<n>P_DEFUALTS in the FFT header file if
// unsure as to what values to program the object with.
CFFT32 cfft = CFFT32_128P_DEFAULTS;

//Calling Functions
GET_DWT_CYCLE_COUNT(dwtCycleCounts[0], CFFT32_brev(ipcbsrc, ipcb, FFT_SIZE));    //
Real part bit-reversing

GET_DWT_CYCLE_COUNT(dwtCycleCounts[1], CFFT32_brev(&ipcbsrc[1], &ipcb[1],
FFT_SIZE)); // Imaginary part bit-reversing
```

6.2. FPU

The floating-point unit (FPU) significantly improves the performance when handling complex mathematical operations. Both Txx320F28004x and G32R501 support FPU, but the former only supports single-precision floating-point arithmetic, while the latter supports both single-precision and double-precision floating-point arithmetic. This section will introduce how to optimize floating-point arithmetic using FPU on G32R501.

The lib file name of R501 FPU library is g32r501_FPU.lib. The FPU library is fully compatible with the FPU library used by Txx320F28004x. The following table is a list of implementation functions of R501 and implementation functions of Txx320F28004x :

Table 102 List of Implementation Functions of R501 FPU and Implementation Functions of Competitive Products

G32R501		Txx320F28004x	
Function name	**Function prototype**	**Function name**	**Function prototype**
FFT Modules			
CFFT_f32	void CFFT_f32(CFFT_F32_STRUCT_Handle hndCFFT_F32)	CFFT_f32	void CFFT_f32(CFFT_F32_STRUCT *);
CFFT_f32t	void CFFT_f32t(CFFT_F32_STRUCT_Handle hndCFFT_F32)	CFFT_f32t	void CFFT_f32t(CFFT_F32_STRUCT *);
CFFT_f32i	void CFFT_f32i(CFFT_F32_STRUCT_Handle hndCFFT_F32)	CFFT_f32i	void CFFT_f32i(CFFT_F32_STRUCT *);
CFFT_f32it	void CFFT_f32it(CFFT_F32_STRUCT_Handle hndCFFT_F32)	CFFT_f32it	void CFFT_f32it(CFFT_F32_STRUCT *);
CFFT_f32u	void CFFT_f32u(CFFT_F32_STRUCT_Handle hndCFFT_F32);	CFFT_f32u	void CFFT_f32u(CFFT_F32_STRUCT *);
CFFT_f32ut	void CFFT_f32ut(CFFT_F32_STRUCT_Handle hndCFFT_F32);	CFFT_f32ut	void CFFT_f32ut(CFFT_F32_STRUCT *);
CFFT_f32_mag	void CFFT_f32_mag(CFFT_F32_STRUCT_Handle hndCFFT_F32)	CFFT_f32_mag	void CFFT_f32_mag(CFFT_F32_STRUCT *);
CFFT_f32_mag_TMU0	void CFFT_f32_mag_TMU0(CFFT_F32_STRUCT_Handle hndCFFT_F32)	CFFT_f32_mag_TMU0	void CFFT_f32_mag_TMU0(CFFT_F32_STRUCT *);
CFFT_f32s_mag	void CFFT_f32s_mag(CFFT_F32_STRUCT_Handle hndCFFT_F32)	CFFT_f32s_mag	void CFFT_f32s_mag(CFFT_F32_STRUCT *);
CFFT_f32s_mag_TMU0	void CFFT_f32s_mag_TMU0(CFFT_F32_STRUCT_Handle hndCFFT_F32)	CFFT_f32s_mag_TMU0	void CFFT_f32s_mag_TMU0(CFFT_F32_STRUCT *);
CFFT_f32_pack	void CFFT_f32_pack(CFFT_F32_STRUCT_Handle hndCFFT_F32)	CFFT_f32_pack	void CFFT_f32_pack(CFFT_F32_STRUCT *);
CFFT_f32_phase	void CFFT_f32_phase(CFFT_F32_STRUCT_Handle hndCFFT_F32)	CFFT_f32_phase	void CFFT_f32_phase(CFFT_F32_STRUCT *);

G32R501		Txx320F28004x	
Function name	Function prototype	Function name	Function prototype
CFFT_f32_phase_TMU0	void CFFT_f32_phase_TMU0(CFFT_F32_STRUCT_Handle hndCFFT_F32)	CFFT_f32_phase_TMU0	void CFFT_f32_phase_TMU0(CFFT_F32_STRUCT *);
CFFT_f32_sincostable	void CFFT_f32_sincostable(CFFT_F32_STRUCT_Handle hndCFFT_F32)	CFFT_f32_sincostable	void CFFT_f32_sincostable(CFFT_F32_STRUCT *);
CFFT_f32_unpack	void CFFT_f32_unpack(CFFT_F32_STRUCT_Handle hndCFFT_F32)	CFFT_f32_unpack	void CFFT_f32_unpack(CFFT_F32_STRUCT *);
void CFFT_f32_win	void CFFT_f32_win(float *pBuffer, const float *pWindow, const uint16_t size)	CFFT32_f32_win	void CFFT32_f32_win(float *, float *, uint16_t);
CFFT_f32_win_dual	void CFFT_f32_win_dual(float *pBuffer, const float *pWindow, const uint16_t size)	CFFT32_f32_win_dual	void CFFT32_f32_win_dual(float *, float *, uint16_t);
ICFFT_f32	void ICFFT_f32(CFFT_F32_STRUCT_Handle hndCFFT_F32)	ICFFT_f32	void ICFFT_f32(CFFT_F32_STRUCT *);
ICFFT_f32t	void ICFFT_f32t(CFFT_F32_STRUCT_Handle hndCFFT_F32)	ICFFT_f32t	void ICFFT_f32t(CFFT_F32_STRUCT *);
RFFT_f32	void RFFT_f32(RFFT_F32_STRUCT_Handle hndRFFT_F32)	RFFT_f32	void RFFT_f32(RFFT_F32_STRUCT *);
RFFT_f32u	void RFFT_f32u(RFFT_F32_STRUCT_Handle hndRFFT_F32);	RFFT_f32u	void RFFT_f32u(RFFT_F32_STRUCT *);
RFFT_adc_f32	void RFFT_adc_f32(RFFT_ADC_F32_STRUCT_Handle hndRFFT_ADC_F32)	RFFT_adc_f32	void RFFT_adc_f32(RFFT_ADC_F32_STRUCT *);
RFFT_adc_f32u	void RFFT_adc_f32u(RFFT_ADC_F32_STRUCT_Handle hndRFFT_ADC_F32);	RFFT_adc_f32u	void RFFT_adc_f32u(RFFT_ADC_F32_STRUCT *);
RFFT_adc_f32_win	void RFFT_adc_f32_win(uint16_t *pBuffer, const uint16_t *pWindow,const uint16_t size)	RFFT_adc_f32_win	void RFFT_adc_f32_win(RFFT_ADC_F32_STRUCT *);
RFFT_f32_mag	void RFFT_f32_mag(RFFT_F32_STRUCT_Handle hndRFFT_F32)	RFFT_f32_mag	void RFFT_f32_mag(RFFT_F32_STRUCT *);

G32R501		Txx320F28004x	
Function name	**Function prototype**	**Function name**	**Function prototype**
RFFT_f32_mag_TMU0	void RFFT_f32_mag_TMU0(RFFT_F32_STRUCT_Handle hndRFFT_F32)	RFFT_f32_mag_TMU0	void RFFT_f32_mag_TMU0(RFFT_F32_STRUCT *);
RFFT_f32s_mag	void RFFT_f32s_mag(RFFT_F32_STRUCT_Handle hndRFFT_F32)	RFFT_f32s_mag	void RFFT_f32s_mag(RFFT_F32_STRUCT *);
RFFT_f32s_mag_TMU0	void RFFT_f32s_mag_TMU0(RFFT_F32_STRUCT_Handle hndRFFT_F32)	RFFT_f32s_mag_TMU0	void RFFT_f32s_mag_TMU0(RFFT_F32_STRUCT *);
RFFT_f32_phase	void RFFT_f32_phase(RFFT_F32_STRUCT_Handle hndRFFT_F32)	RFFT_f32_phase	void RFFT_f32_phase(RFFT_F32_STRUCT *);
RFFT_f32_phase_TMU0	void RFFT_f32_phase_TMU0(RFFT_F32_STRUCT_Handle hndRFFT_F32)	RFFT_f32_phase_TMU0	void RFFT_f32_phase_TMU0(RFFT_F32_STRUCT *);
RFFT_f32_sincostable	void RFFT_f32_sincostable(RFFT_F32_STRUCT_Handle hndRFFT_F32)	RFFT_f32_sincostable	void RFFT_f32_sincostable(RFFT_F32_STRUCT *);
RFFT_f32_win	void RFFT_f32_win(float *pBuffer, const float *pWindow, const uint16_t size)	RFFT_f32_win	void RFFT_f32_win(float *, float *, uint16_t)
Filter Modules			
FIR_f32_calc	void FIR_f32_calc(FIR_f32_Handle hndFIR_f32)	FIR_f32_calc	void FIR_f32_calc(FIR_f32_handle);
IIR_f32_calc	void IIR_f32_calc(IIR_f32_Handle hndIIR_f32)	IIR_f32_calc	void IIR_f32_calc(IIR_f32_handle);
Vector Modules			
abs_SP_CV	void abs_SP_CV(float *y, const complex_float *x, const uint16_t N)	abs_SP_CV	void abs_SP_CV(float32 *, const complex_float *, const Uint16);
abs_SP_CV_2	void abs_SP_CV_2(float *y, const complex_float *x, const uint16_t N)	abs_SP_CV_2	void abs_SP_CV_2(float32 *, const complex_float *, const Uint16);
abs_SP_CV_TMU0	void abs_SP_CV_TMU0(float *y, const complex_float *x, const uint16_t N)	abs_SP_CV_TMU0	void abs_SP_CV_TMU0(float32 *, const complex_float *, const Uint16);
add_SP_CSxCV	void add_SP_CSxCV(complex_float *y, const complex_float *x, const complex_float c, const uint16_t N)	add_SP_CSxCV	void add_SP_CSxCV(complex_float *, const complex_float *,

G32R501		Txx320F28004x	
Function name	Function prototype	Function name	Function prototype
			const complex_float, const Uint16);
add_SP_CVxCV	void add_SP_CVxCV(complex_float *y, const complex_float *w,const complex_float *x, const uint16_t N)	add_SP_CVxCV	void add_SP_CVxCV(complex_float *, const complex_float *, const complex_float *, const Uint16);
iabs_SP_CV	void iabs_SP_CV(float *y, const complex_float *x, const uint16_t N)	iabs_SP_CV	void iabs_SP_CV(float32 *, const complex_float *, const Uint16);
iabs_SP_CV_2	void iabs_SP_CV_2(float *y, const complex_float *x, const uint16_t N)	iabs_SP_CV_2	void iabs_SP_CV_2(float32 *, const complex_float *, const Uint16);
iabs_SP_CV_TMU0	void iabs_SP_CV_TMU0(float *y, const complex_float *x, const uint16_t N)	iabs_SP_CV_TMU0	void iabs_SP_CV_TMU0(float32 *, const complex_float *, const Uint16);
mac_SP_CVxCV	complex_float mac_SP_CVxCV(const complex_float *w,const complex_float *x, const uint16_t N)	mac_SP_CVxCV	complex_float mac_SP_RVxCV(const complex_float *, const complex_float *, const uint16_t);
mac_SP_i16RVxCV	complex_float mac_SP_i16RVxCV(const complex_float *w, const int16_t *x,const uint16_t N)	mac_SP_RVxCV	complex_float mac_SP_RVxCV(const complex_float *, const float *, const uint16_t);
mac_SP_RVxCV	complex_float mac_SP_RVxCV(const complex_float *w,const float *x, const uint16_t N)	mac_SP_i16RVxCV	complex_float mac_SP_i16RVxCV(const complex_float *, const int16_t *, const uint16_t);
maxidx_SP_RV_2	uint16_t maxidx_SP_RV_2(const float *x, const uint16_t N)	maxidx_SP_RV_2	Uint16 maxidx_SP_RV_2(float32 *, Uint16);
mean_SP_CV_2	complex_float mean_SP_CV_2(const complex_float *x, const uint16_t N)	mean_SP_CV_2	complex_float mean_SP_CV_2(const complex_float *, const Uint16);
median_noreorder_SP_RV	float median_noreorder_SP_RV(const float *x, const uint16_t N)	median_noreorder_SP_RV	float32 median_noreorder_SP_RV(const float32 *, Uint16);
median_SP_RV	float median_SP_RV(float *x, const uint16_t N)	median_SP_RV	float32 median_SP_RV(float32 *, Uint16);

G32R501		Txx320F28004x	
Function name	Function prototype	Function name	Function prototype
mpy_SP_CSxCS	complex_float mpy_SP_CSxCS(const complex_float w, const complex_float x)	mpy_SP_CSxCS	complex_float mpy_SP_CSxCS(complex_float, complex_float);
mpy_SP_CVxCV	void mpy_SP_CVxCV(complex_float *y, const complex_float *w, const complex_float *x, const uint16_t N)	mpy_SP_CVxCV	void mpy_SP_CVxCV(complex_float *, const complex_float *, const complex_float *, const Uint16);
mpy_SP_CVxCVC	void mpy_SP_CVxCVC(complex_float *y, const complex_float *w,const complex_float *x, const uint16_t N)	mpy_SP_CVxCVC	void mpy_SP_CVxCVC(complex_float *, const complex_float *, const complex_float *, const Uint16);
mpy_SP_RSxRV_2	void mpy_SP_RSxRV_2(float *y, const float *x, const float c,const uint16_t N)	mpy_SP_RSxRV_2	void mpy_SP_RSxRV_2(float32 *, const float32 *, const float32, const Uint16);
mpy_SP_RSxRVxRV_2	void mpy_SP_RSxRVxRV_2(float *y, const float *w, const float *x,const float c, const uint16_t N)	mpy_SP_RSxRVxRV_2	void mpy_SP_RSxRVxRV_2(float32 *, const float32 *, const float32 *, const float32, const Uint16);
mpy_SP_RVxCV	void mpy_SP_RVxCV(complex_float *y, const complex_float *w,const float *x, const uint16_t N)	mpy_SP_RVxCV	void mpy_SP_RVxCV(complex_float *, const complex_float *, const float32 *, const Uint16);
mpy_SP_RVxRV_2	void mpy_SP_RVxRV_2(float *y, const float *w, const float *x,const uint16_t N)	mpy_SP_RVxRV_2	void mpy_SP_RVxRV_2(float32 *, const float32 *, const float32 *, const Uint16);
qsort_SP_RV	void qsort_SP_RV(void *x, const uint16_t N)	qsort_SP_RV	void qsort_SP_RV(void *, Uint16);
rnd_SP_RS	float rnd_SP_RS(const float x)	rnd_SP_RS	float32 rnd_SP_RS(float32);
sub_SP_CSxCV	void sub_SP_CSxCV(complex_float *y, const complex_float *x,const complex_float c, const uint16_t N)	sub_SP_CSxCV	void sub_SP_CSxCV(complex_float *, const complex_float *, const complex_float, const Uint16);
sub_SP_CVxCV	void sub_SP_CVxCV(complex_float *y, const complex_float *w,const complex_float *x, const uint16_t N)	sub_SP_CVxCV	void sub_SP_CVxCV(complex_float *, const complex_float *,

G32R501		Txx320F28004x	
Function name	Function prototype	Function name	Function prototype
			const complex_float *, const Uint16);

The fpu_fft.h file contains the declaration of implementation functions for the FPU library and definition of related structure parameters. Users can port the use cases of FPU library functions used on Txx320F28004x to G32R501 according to the implementation function list.

6.3. VCU

The vector computing unit (VCU) is commonly used for efficient processing of vector operations. G32R501 implements similar function through DSP instruction set and Helium support.

6.3.1. VCU0 libraries

The lib file name of R501 VCU library is g32r501_VCU.lib. The VCU library is compatible with the FPU library used by Txx320F28004x. The following table is a list of implementation functions of R501 and implementation functions of Txx320F28004x :

Table 103 List of Implementation Functions of R501 VCU and Implementation Functions of Txx320F28004x

G32R501		Txx320F28004x	
Function name	Function name	Function name	Function prototype
Fast Fourier Transform			
cfft16_128p_calc	void cfft16_128p_calc(cfft16_t *cfft16_handle_s)	cfft16_128p_calc	void cfft16_128p_calc (cfft16_t *cfft16_handle_s)
cfft16_256p_calc	void cfft16_256p_calc(cfft16_t *cfft16_handle_s)	cfft16_256p_calc	void *cfft16_256p*_calc (cfft16_t *cfft16_handle_s)
cfft16_64p_calc	void cfft16_64p_calc(cfft16_t *cfft16_handle_s)	cfft16_64p_calc	void *cfft16_64p*_calc (cfft16_t *cfft16_handle_s)
cfft16_brev	void cfft16_brev(cfft16_t *cfft16_handle_s)	cfft16_brev	void *cfft16*_brev (cfft16_t * cfft16_handle_s)

cfft16_flip_re_img	void cfft16_flip_re_img(cfft16_t *cfft16_handle_s)	cfft16_flip_re_img	void *cfft16_flip_re*_img (cfft16_t ∗cfft16_handle_s)
cfft16_flip_re_img_conj	void cfft16_flip_re_img_conj(cfft16_t *cfft16_handle_s)	cfft16_flip_re_img_conj	void *cfft16_flip_re*_img_conj (cfft16_t ∗cfft16_handle_s)
cfft16_init	void cfft16_init(cfft16_t *cfft16_handle_s)	cfft16_init	void *cfft16*_init (cfft16_t ∗ cfft16_handle_s)
cfft16_unpack	void cfft16_unpack(cfft16_t *cfft16_handle_s)	cfft16_unpack_asm	void *cfft16_unpack*_asm (cfft16_t ∗cfft16_handle_s)
cifft16_pack	void cifft16_pack(cfft16_t *cfft16_handle_s)	cifft16_pack_asm	void *cifft16_pack*_asm (cfft16_t ∗cfft16_handle_s)
Cyclic Redundancy Check			
CRC_reset	void CRC_reset(void)	CRC_reset	void CRC_reset (void)
-	-	flipInputBuf_cpu	void flipInputBuf_cpu (uint16 ∗dst, uint16 ∗src, uint16 rxLen)
genCRC16P1Table	void genCRC16P1Table(void)	genCRC16P1Table	void genCRC16P1Table ()
genCRC16P2Table	void genCRC16P2Table(void)	genCRC16P2Table	void genCRC16P2Table ()
genCRC32Table	void genCRC32Table(void)	genCRC32Table	void genCRC32Table ()
genCRC8Table	void genCRC8Table(void)	genCRC8Table	void genCRC8Table ()
getCRC16P1_cpu	uint16 getCRC16P1_cpu (uint16 input_crc16_accum, uint16 * msg,	getCRC16P1_cpu	uint16 getCRC16P1_cpu (uint16 input_crc16_accum, uint16

	CRC_parity_e parity,uint16 rxLen)		*msg, CRC_parity_e parity,uint16 rxLen)
getCRC16P1_vcu	uint16 getCRC16P1_vcu(uint32 input_crc16_accum, uint16 * msg, CRC_parity_e parity,uint16 rxLen)	getCRC16P1_vcu	uint16 getCRC16P1_vcu (uint32 input_crc16_accum, uint16 *msg, CRC_parity_e parity,uint16 rxLen)
getCRC16P2_cpu	uint16 getCRC16P2_cpu (uint16 input_crc16_accum, uint16 * msg, CRC_parity_e parity,uint16 rxLen)	getCRC16P2_cpu	uint16 getCRC16P2_cpu (uint16 input_crc16_accum, uint16 *msg, CRC_parity_e parity,uint16 rxLen)
getCRC16P2_vcu	uint16 getCRC16P2_vcu(uint32 input_crc16_accum, uint16 * msg, CRC_parity_e parity,uint16 rxLen)	getCRC16P2_vcu	uint16 getCRC16P2_vcu (uint32 input_crc16_accum, uint16 *msg, CRC_parity_e parity,uint16 rxLen)
getCRC32_cpu	uint32 getCRC32_cpu (uint32 input_crc32_accum, uint16 * msg, CRC_parity_e parity,uint16 rxLen)	getCRC32_cpu	uint32 getCRC32_cpu (uint32 input_crc32_accum, uint16 *msg, CRC_parity_e parity, uint16 rxLen)
getCRC32_vcu	uint32 getCRC32_vcu(uint32 input_crc32_accum, uint16 *msg, CRC_parity_e parity, uint16 rxLen)	getCRC32_vcu	uint32 getCRC32_vcu (uint32 input_crc32_accum, uint16 *msg, CRC_parity_e parity, uint16 rxLen)
-	-	getCRC32_vcu_hilo_order_swap	uint32 getCRC32_vcu_hilo_order_swap (uint32 input_crc32_accum, uint16

			∗msg, CRC_parity_e parity, uint16 rxLen)
getCRC8_cpu	uint16 getCRC8_cpu (uint16 input_crc8_accum, uint16 * msg, CRC_parity_e parity, uint16 rxLen)	getCRC8_cpu	uint16 getCRC8_cpu (uint16 input_crc8_accum, uint16 ∗msg, CRC_parity_e parity, uint16 rxLen)
getCRC8_vcu	uint8 getCRC8_vcu(uint32 input_crc8_accum, uint16 *msg, CRC_parity_e parity, uint16 rxLen)	getCRC8_vcu	uint16 getCRC8_vcu (uint32 input_crc8_accum, uint16 ∗msg, CRC_parity_e parity, uint16 rxLen)
Viterbi Decoding (VCU0)			
cnvDec	void cnvDec(int16 nBits, int16 *in_p, int16 *out_p, int16 flag)	cnvDec_asm	void cnvDec_asm (int nBits, int ∗in_p, int ∗out_p, int flag)
cnvDecInit	void cnvDecInit(int16 nTranBits)	cnvDecInit_asm	void cnvDecInit_asm (int nTranBits)
cnvDecMetricRescale	void cnvDecMetricRescale(void)	cnvDecMetricRescale_asm	void cnvDecMetricRescale_asm ()

Description for functions with different names and unimplemented functions of G32R501 and Txx320F28004x:

- For the functions cfft16_unpack, cifft16_pack, cnvDec, cnvDecInit and cnvDecMetricRescale, only their names are changed, but their parameters and functions are not changed.

- Function: flipInputBuf_cpu

  Description: This function is used to flip a 16-bit input buffer.

  If the user needs to use bit flipping during FFT calculation, R501 VCU0 provides the function void cfft16_brev(cfft16_t *cfft16_handle_s), and the cfft16_brev function implements a bit flipping operation for a 16-point complex FFT. Through this operation, the data in the input buffer can be rearranged for subsequent calculation of the FFT algorithm.

- Function: getCRC32_vcu_hilo_order_swap

  Description: This function calculates the 32-bit CRC of the message buffer.

  The getCRC32_vcu function in the R501 VCU0 library implements the function of calculating 32-bit CRC using the VCU instruction.

The vcu0_fft.h file contains the declaration for the implementation functions of the VCU0 library and the definitions of related structure parameters. Compared with related files of the Txx320F28004x VCU0 library, basically R501 can be fully compatible with the Txx320F28004x VCU0 library.

6.4. Fix32math (benchmarking IQmath)

The Fix32math library benchmarks the Txx320F28004x IQmath library for efficient implementation of fixed-point mathematical operations.

The lib file name of R501 Fix32math library is g32r501_Fix32math.lib, and it is fully compatible with the IQmath library used by Txx320F28004x. The following table is a list of implementation functions of R501 and implementation functions of Txx320F28004x :

Table 104 List of Implementation Functions of R501 IQmath Library and Implementation Functions of Txx320F28004x

G32R501	Txx320F28004x
Function name	**Function name**
Conversion Utilities	
IQN	IQN
IQNtoF	IQNtoF
atoIQN	atoIQN
IQNtoa	IQNtoa
IQNint	IQNint
IQNfrac	IQNfrac
IQtoIQN	IQtoIQN
IQNtoIQ	IQNtoIQ
IQtoQN	IQtoQN
QNtoIQ	QNtoIQ
Shift to Multiply or Divide by Powers of 2	
IQmpy2,4,8…64	IQmpy2,4,8…64
IQdiv2,4,8…64	IQdiv2,4,8…64
Arithmetic Operations	
IQNmpy	IQNmpy
IQNrmpy	IQNrmpy
IQnrsmpy	IQnrsmpy
IQNmpyI32	IQNmpyI32

G32R501	Txx320F28004x
Function name	**Function name**
IQNmpyIQX	IQNmpyIQX
IQNdiv	IQNdiv
Trigonometric Functions	
IQNasin	IQNasin
IQNsin	IQNsin
IQNsinPU	IQNsinPU
IQNacos	IQNacos
IQNcos	IQNcos
IQNcosPU	IQNcosPU
IQNatan2	IQNatan2
IQNatan2PU	IQNatan2PU
IQNatan	IQNatan
Mathematical Utilities	
IQNexp	IQNexp
IQNlog	IQNlog
IQNsqrt	IQNsqrt
IQNisqrt	IQNisqrt
IQNmag	IQNmag
Miscellaneous Utilities	
IQNabs	IQNabs
IQsat	IQsat

Each function in the Fix32math library has IQ0~IQ30, 30 IQ formats. Taking IQN as an example. In the global IQ function (IQ format=GLOBAL_Q, GLOBAL_Q is defined in the file Fix32mathLib.h), the calling method of the function is _iq _IQ(float F); the calling method of the specific IQ function for IQ format (IQ format=IQ1 to IQ29) is _iqN _IQN(float F)

The declaration for IQ function is made in the file Fix32mathLib.h. Compared with relevant files of the Txx320F28004x IQmath library, R501 is fully compatible with the IQmath library of Txx320F28004x, and the Fix32math library is fully compatible with the IQmath library of Txx320F28004x.

6.5. FPUfastRTS

The FPUfastRTS library is used to quickly implement floating-point arithmetic.

The lib file name of R501 FPUfastRTS library is g32r501_FPUfastRTS.lib, and it is fully compatible with the FPUfastRTS library used by Txx320F28004x. The following table is a list of implementation functions of R501 and implementation functions of Txx320F28004x:

Table 105 List of Implementation Functions of R501 FPUfastRTS Library and Implementation Functions of Txx320F28004x

G32R501		Txx320F28004x	
Function name	**Function name**	**Function name**	**Function prototype**
Arithmetic and Trigonometric			
atan2f	float32_t atan2f_1(float32_t Y, float32_t X);	atan2f	float32_t atan2f (float32_t Y, float32_t X);
atanf	float32_t atanf_1(float32_t X);	atanf	float32_t atanf (float32_t X);
cosf	float32_t cosf_1(float32_t X);	cosf	float32_t cosf (float32_t X);
expf	float32_t expf_1 (float32_t X);	expf	float32_t expf (float32_t X);
-	-	FS$$DIV	float32_t FS $$DIV (float32_t X, float32_t Y);
isqrtf	float32_t isqrtf_1(float32_t X);	isqrtf	float32_t isqrtf (float32_t X);
logf	float32_t logf_1(float32_t X);	logf	float32_t logf (float32_t X);
powf	float32_t powf_1(float32_t X, float32_t Y);	powf	float32_t powf (float32_t X, float32_t Y);
sincosf	void sincosf_1(float32_t radian, float32_t *PtrSin, float32_t *PtrCos);	sincosf	void sincosf (float32_t radian, float32_t *PtrSin, float32_t *PtrCos);
sinf	float32_t sinf_1(float32_t X);	sinf	float32_t sinf (float32_t X);
sqrtf	float32_t sqrtf_1 (float32_t X);	sqrtf	float32_t sqrtf (float32_t X);
acosf	float32_t acosf_1(float32_t X);	acosf	float32_t acosf (float32_t X)
asinf	float32_t asinf_1(float32_t X);	asinf	float32_t asinf (float32_t X)

6.6. Zidian

To extend the functions of R501, a Zidian module is added to R501, and it provides hardware acceleration for certain specific operations, but there is no direct corresponding module in Txx320F28004x. R501 implements the ability to accelerate the execution of common trigonometric functions and arithmetic operations through customized instruction sets and hardware acceleration unit.

The instructions in Zidian include two categories, i.e. CAU and FACU:

- ICAU: Improve the performance of integer operations, including operations requiring Helium calculation: FFT, Complex operation, etc.; CRC algorithm.
- FACU: Improve the performance of floating-point operations, including trigonometric calculation, square root, and division.

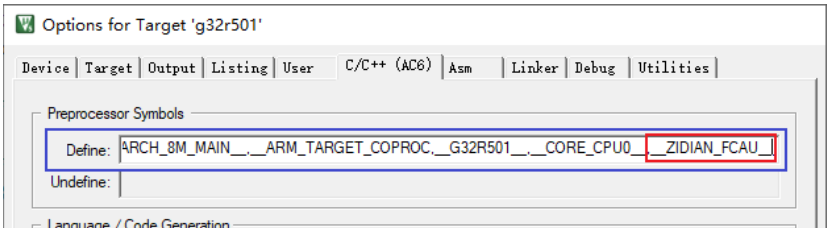
In Txx320F28004x, select whether to enable the TMU function through **--tmu_support=tmu0** to perform calculations using the TMU variants of certain functions in FPU and FPUfastRTS libraries.

All arithmetic instructions in Txx320F28004x TMU can be replaced by the ICAU and FCAU instructions or the basic instructions and DSP instructions of Arm® Cortex®-M52 processors. More than one ARM instruction (CDE (Custom Datapath Extension) or ARM native instruction) may be required to replace one TMU instruction.

Note: CDE (Custom Datapath Extension) is a feature of Arm® Cortex®-M52 processor. The designers are allowed to add custom hardware acceleration function to the standard processor architecture. This extension function allows developers to optimize the processor performance for specific applications and implement more efficient processing capability.

If users need to use a function similar to the TMU variant function in Txx320F28004x, they just need to add __ZIDIAN_FCAU__ to the Define under C/C++ tab in option for target, to enable the Zidian function, and then call the required calculation function. The following example will demonstrate how to apply Zidian to use the TMU variant function in G32R501 library.

Figure 2 Add __ZIDIAN_FCAU__ Macro Definition in keil Configuration Interface



Example 1:

```
#ifdef __ZIDIAN_FCAU__
#define TMU
#endif
......

#ifdef TMU       // when defined __ZIDIAN_FCAU__ in the project options
    // Run the calculation function and measure the DWT cycle count simultaneously
    // Calculate magnitude, result stored in CurrentOutPtr
    GET_DWT_CYCLE_COUNT(dwtCycleCounts[3],CFFT_f32_mag_TMU0(hnd_cfft));
#else
    // Run the calculation function and measure the DWT cycle count simultaneously
    // Calculate magnitude, result stored in CurrentOutPtr
    GET_DWT_CYCLE_COUNT(dwtCycleCounts[3],CFFT_f32_mag(hnd_cfft));

#endif
```

This example code demonstrates the use of the CFFT_f32_mag_TMU0 function in the FPU library by adding the macro __ZIDIAN_FCAU__.

Example 2:

```
GET_DWT_CYCLE_COUNT(dwtCycleCounts[0], out.f32    = atanf(in.f32));
```

This example code demonstrates direct calling of corresponding functions in the FPUfastRTS library by adding the macro __ZIDIAN_FCAU__, to implement Zidian acceleration operations.

For more information on the use of Zidian, please refer to the *G32R5xx Zidian Application Notes*.

Note: In the FPUfastRTS library function table, expf, logf, and powf do not implement Zidian acceleration function.

7. Programming Mode Porting

Different microcontroller architectures have significant differences in instruction sets, interrupt handling mechanisms, register protection, and other aspects, so special attention needs to be paid to the compatibility and adaptation of these programming modes in the migration process.

7.1. Dual-core Situation

The G32R501 of dual-core version integrates two Arm® Cortex®-M52 cores, which can work in parallel to perform high-performance computing tasks and real-time control tasks at the same time.

7.1.1. Start CPU1

CPU0 starts by default when powered on. When users need to start CPU1, they need to set the BOOTCTRL1 register to 0x2 in the program to enable CPU1's clock and start CPU1.

7.1.2. Dual-core communication

CPU0 and CPU1 can exchange data and communicate through the internal inter-processor communication unit (IPC) and the shared RAM storage area.

7.1.3. Resource competition

CPU0 and CPU1 share the same Flash **storage space**. Therefore, when designing the dual-core applications, it is necessary to reasonably allocate the Flash storage space occupied by CPU0 and CPU1. In the dual-core example provided in G32R5xx SDK, the Flash storage space is evenly divided for two CPU cores. If users want to increase the Flash storage space occupied by CPU0 or CPU1, it can be implemented by modifying the link file provided in the SDK example.

In addition, **all peripheral resources** and **SRAM1-3 storage area** can be accessed by CPU0 and CPU1. Since G32R501 has no hardware mutual exclusion mechanism, if two CPU cores use the same peripheral or SRAM storage area, there will be resource competition.

The following is an example and description of two CPU running the "GPIO Flip" program simultaneously, which can be used as a reference in practical applications:

In dual-bank mode, CPU0 executes programs in Flash Bank0, and CPU1 also executes programs in Flash Bank0. When CPU0 is executing a program, the GPIO flip time of CPU1 will increase.

- When CPU0 and CPU1 flip IO simultaneously, the dual cores will access the GPIO peripherals simultaneously, which will cause competition on the bus. Since the CPU0 has a higher priority, the cycle of CPU1 flipping IO will become longer.
- In dual-Bank mode, the dual cores simultaneously access a FLASH; in single-Bank mode,

when the dual cores access FLASH simultaneously, there will be bus competition. Since the CPU0 has a higher priority, CPU0 will fetch data from FLASH, and CPU1 fetching instructions from FLASH will be affected by bus competition.

The dual-core example ipc_ex4_resource_share provided by SDK demonstrates how to use the GP (general-purpose) interrupt of the IPC module to achieve synchronous and mutually exclusive access in case of resource competition. For detailed introduction, please refer to the document *AN1138_G32R501 IPC Application Notes*.

7.1.4. CLA functions

In Txx320F28004x, CLA (Control Law Accelerator) is a coprocessor, which is designed to handle real-time control tasks. It can perform independent computing tasks independent of the main CPU, mainly used to accelerate the control algorithms such as PID controllers and filters, which can significantly improve the real-time performance of the system. CLA can directly access on-chip RAM and peripheral registers, perform floating-point and fixed-point arithmetic, and is suitable for control applications that require fast response.

If CLA of Txx320F28004x is used in applications, CPU1 can be used in G32R501 to assume a corresponding role. Specifically:

- Computational tasks: CPU1 can independently perform computational tasks, including function operations and trigonometric operations.
- Dual-core communication: Efficient communication between dual cores can be achieved through IPC (Inter-Process Communication) or shared memory area.

7.2. Interrupt Handling

There are significant differences in interrupt controllers and interrupt processing methods between Txx320F28004x and G32R501 microcontrollers. Txx320F28004x adopts PIE (Peripheral Interrupt Expansion) mechanism, while G32R501 integrates Nested Vectored Interrupt Controller (NVIC). In the migration process, it is necessary to reconfigure the interrupt vector table and interrupt priority in order to adapt to the new interrupt processing mechanism.

For interrupt processing of G32R501 microcontroller, in addition to the content of this chapter, please refer to the document *Arm China Cortex®-M52 Processor Technical Reference Manual*.

7.2.1. Interrupt nesting

G32R501 adopts NVIC (Nested Vectored Interrupt Controller) mechanism, which supports interrupt nesting, and the high-priority interrupts are allowed to interrupt the low-priority interrupt service routine (ISR). Its characteristics include:

- G32R501 provides up to 226 interrupt vectors, each with an independent priority.
- Interrupt priority is divided into preempt priority and sub-priority, with high flexibility, so that

the developers can finely control the response sequence of interrupts.

- G32R501 has programmable interrupt priority, which depends on the specific implementation scheme (usually 4 bits). The developers can configure interrupt priority in applications in order to control the preemption behavior of interrupts.

Txx320F28004x adopts PIE (Peripheral Interrupt Expansion) mechanism, and the interrupt priority setting logic is implemented through grouping and channels, and does not support interrupt nesting.

7.2.2. Interrupt priority

The interrupt control process of Txx320F28004x is as follows:

- Interrupt grouping: PIE divides interrupts into 12 groups, each containing 8 channels. For example, PIE Group 1 is responsible for processing the interrupt requests from Peripheral 1 to Peripheral 8.
- Priority setting: Within each group, the priority of each channel can be set. The priority is set through PIEIER (PIE interrupt enable register) and PIEIFR (PIE interrupt flag register).

  ```
  PieCtrlRegs.PIEIERx.bit.INTy = 1;
  ```

For G32R501 interrupt priority setting, first configure the priority grouping, and then set specific preempt priority and sub priority for each interrupt source.

- **Priority grouping:** Before setting the priority of specific interrupt, it is necessary to configure the priority grouping of the entire chip. The priority grouping defines the ratio of preempt priority and sub priority.

  Priority grouping can be set through the following function:

  ```
  Interrupt_setPriorityGroup(uint32_t priorityGroup);
  ```

  For example, set the priority grouping to 7-bit preempt priority and 1-bit sub priority:

  ```
  Interrupt_setPriorityGroup(NVIC_PRIORITYGROUP_7_1);
  ```

- **Priority setting:** Through the Interrupt_setPriority function, the priority can be set for each interrupt source, including preempt priority and sub priority. Users can flexibly configure the priority of the current interrupt to the specific priority group and priority level.

  ```
  Interrupt_setPriority(int32_t interruptNumber, uint32_t preemptPriority,
                        uint32_t subPriority)
  ```

  For example, set the priority of a certain peripheral interrupt:

  ```
  Interrupt_setPriority(INT_XINT1, 0, 0);
  ```

7.2.3. Interrupt enable

In Txx320F28004x, the interrupt controller PIE (Peripheral Interrupt Expansion) needs to be configured through the enable bit ENPIE:

```
PieCtrlRegs.PIECTRL.bit.ENPIE = 1;
```

In G32R501, interrupt enable is configured through NVIC. The following function can be used to enable specific interrupts:

```
Interrupt_enable(int32_t interruptNumber);
```

For example, enabling a certain peripheral interrupt:

```
Interrupt_enable(INT_XINT1);
```

7.2.4. Interrupt exit

In Txx320F28004x, when the interrupt service program is over, it is necessary to explicitly clear the ACK (Acknowledge) flag bit to notify the PIE controller that the interrupt processing has been completed:

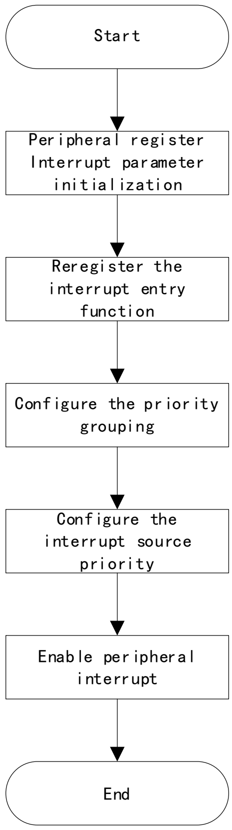
```
PieCtrlRegs.PIEACK.all = PIEACK_GROUPx;
```

In G32R501, there is no need to explicitly clear the interrupt suspension state when the interrupt service program is over, and NVIC will automatically process it. Once the interrupt service program has been executed, the return operation will automatically clear the interrupt suspension state.

7.2.5. Interrupt initialization process

In G32R501, if external interrupt function is required, for the process configuration, refer to Figure 3.

Figure 3 Interrupt Initialization Process



The code reference is as follows:

```
//
// Initializes NVIC and clears NVIC registers. Disables CPU interrupts.
// and clear all CPU interrupt flags.
//
Interrupt_initModule();

//
// Initializes the NVIC vector table with pointers to the shell Interrupt
// Service Routines (ISR).
//
Interrupt_initVectorTable();

//
// Set the priority group to indicate the PREEMPT and SUB priority bits.
//
Interrupt_setPriorityGroup(INTERRUPT_PRIGROUP_PREEMPT_7_6_SUB_5_0);

//
```

```
// Set the global and group interrupt priority to allow CPU interrupts
// with higher priority
Interrupt_setPriority(INT_XINT1, 0, 0);

//
// Board Initialization
//
Board_init();

GPIO_setInterruptType(GPIO_INT_XINT1, GPIO_INT_TYPE_FALLING_EDGE);
GPIO_setInterruptPin(myGPIOInputInterrupt0, GPIO_INT_XINT1);
GPIO_enableInterrupt(GPIO_INT_XINT1);

Interrupt_register(INT_XINT1, &gpioInterruptHandler);
Interrupt_enable(INT_XINT1);

//
// Enables CPU interrupts
//
Interrupt_enableMaster();
```

7.3. RPT Instruction

In the instruction set of Txx320F28004x, there is an instruction called RPT (Repeat), which is commonly used to implement the delay function. For example:

```
asm(" RPT #5 || NOP ");
```

This instruction represents repeating the NOP operation 5 times to implement precise short time delay.

In the G32R501 series chip, Arm® Cortex®-M52 core is used. Arm® Cortex®-M52 core does not directly support the instructions similar to RPT. Therefore, it is recommended to use the following methods to implement the similar delay function.

Use the NOP instruction for control: In Arm® Cortex®-M52 core, similar delay effect can be produced by writing custom delay function. The following is an example of a function called SysCtl_delay, which implements loop delay through assembly code:

```
void SysCtl_delay(uint32_t count)
{
    __asm volatile
    (
                                        "MOV R1, LR \n"
                                        "MOV LR, R0 \n"
```

```
            "delay_loop: \n"
                                        "NOP \n"
                                        "NOP \n"
                                        "NOP \n"
                                        "NOP \n"
                                          "SUBS LR, LR, #1 \n"
                                        "BNE delay_loop \n"
             "delay_end: \n"
                                        "NOP \n"
                                        "NOP \n"
                                        "MOV LR, R1 \n"
    );
}
```

Although the Arm® Cortex®-M52 core of G32R501 series chips does not support direct RPT instruction, similar delay effect can still be produced by writing custom delay functions such as the SysCtl_delay function. Users can adjust the number of delay periods according to the actual needs so as to reach the desired delay length.

7.4. Register Access

The register access of G32R501 series chips needs to follow specific rules, including offset address and access bit width, and some registers have the write protection mechanism. When performing write operation, it is necessary to first remove the write protection, and after the operation is completed, re-enable the write protection. Users shall carefully refer to the *User Manual* in the development process to ensure correct access and operation of registers.

G32R501 supports 8-bit addressing, but the following rules must be followed when accessing registers:

Offset address:

- The register offset address of G32R501 differs from Txx320F28004x. Please refer to the *User Manual* for specific information.

Access rules:

- If the bit width of the register is 16 bits, it must be accessed according to 16 bits in G32R501.
- If the bit width of the register is 32 bits, it must be accessed according to 32 bits in G32R501.

Write register protection (WRPRT):

- The WRPRT mechanism of G32R501 is designed to prevent CPU from performing erroneous write operations on some registers in the system. This mechanism uses special co-processor instructions ("WRPRT" and "WRPRTDIS") to enable and disable access to protected registers.

The User Manual indicates that the ADCCTL1 register of the ADC module has the write protection mechanism. When write operations is performed on this register, the write protection needs to be released first, and after the operation is completed, the write protection shall be re-enabled. The example codes are as follows:

```
//
// Set the position of the pulse.
//
WRPRT_ENABLE;
HWREGH(ADCA_BASE + ADC_O_CTL1) =
(HWREGH(ADCA_BASE + ADC_O_CTL1) & ~ADC_CTL1_INTPULSEPOS) |
(uint16_t)ADC_PULSE_END_OF_ACQ_WIN;
WRPRT_DISABLE;
```

8. Simulation Support

In embedded development, the use of simulation is crucial. The behavior of different chips in simulation mode may vary, especially in terms of interrupt processing and breakpoint settings. The following is detailed discussion and recommendation on simulation support compatibility of Txx320F28004x series and G32R501 series MCU.

8.1. Hardware Support

Compared to Txx320F28004x, G32R501 not only supports JTAG interface, but also supports SWD interface for simulation.

Support of simulators by G32R501:

- Geehy-Link (WinUSB), DAP Link (the firmware version is CMSIS-DAP V2 and above)
- J-Link V12 (J-Link V7.94g and above)
- Ulink Pro

8.2. Interrupt Response Behavior

The behavior of interrupt processing during simulation debugging has a significant impact on software development and debugging. There are significant differences between Txx320F28004x series and G32R501 series in this aspect.

Txx320F28004x series

Behavior: When setting breakpoints in the main function, the interrupts can still respond normally. This behavior allows developers to use interrupts to detect and handle events during debugging, without affecting the execution of the main program.

G32R501 series

Behavior: When setting breakpoints in the main function, usually all CPU activities will be stopped, including interrupt response. This behavior makes it impossible to trigger and process interrupts during the pause period of breakpoints.

The suggestions for the solutions to this difference in the G32R501 series are as follows:

- Understand and accept the limitation that the interrupt cannot respond in breakpoint state, and adjust the debugging strategy.

8.3. Dual-core Simulation

Different from the Txx320F28004x, the G32R501 series MCU has a dual-core system. When making dual-core simulation, special attention shall be paid to debugging and synchronization of the two CPU cores.

Implement the debugging process for CPU1:

- Boot CPU0: First boot CPU0 and execute the BootROM code.
- Jump to application code: CPU0 jumps to application code after BootROM is executed.
- Enable CPU1 clock: In the application code, set the value of BOOTCTRL1 register to 0x2 to enable the CPU1 clock.
- Connect and debug CPU1: After the above steps are completed, CPU1 can be connected and debugged normally.

8.3.1. Dual-core debug system (DCDS)

The simulation system used by the G32R501 series MCU is a dual-core debug system. To know how to conduct dual-core simulation, please refer to the AN1128_G32R501 Dual-core Emulation Guide.

Figure 4 Structure Block Diagram

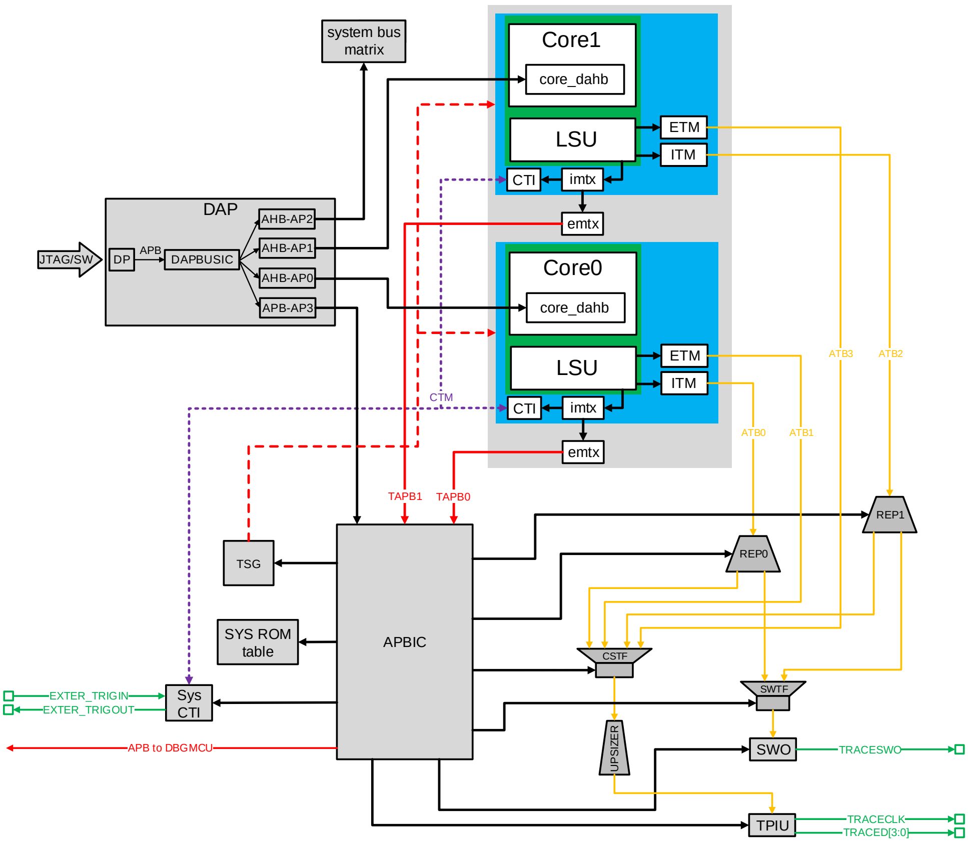
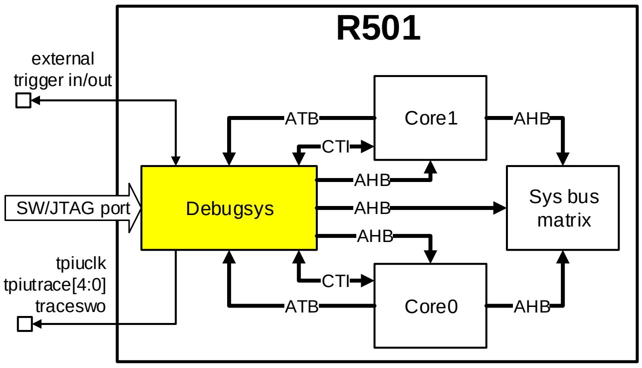


Figure 5 Typical Integration of Debug System



Main characteristics:

- Support independent breakpoint debugging for each CPU core in the system
- Support code execution tracking
- Support JTAG debug port
- Support serial line debug port
- Support software instrumentation
- Support cross triggering
- Support trigger input and trigger output
- Support access to the system bus matrix
- Support serial line (SWO) tracking port
- Support TPIU tracking port

9. Keil MDK-ARM Development Tool Chain

During the migration from Txx320F28004x series MCU to G32R501 series MCU, the porting of development tools is an important link. Code Composer Studio (CCStudio) integrated development environment (IDE) and Keil MDK-ARM are both commonly used tools in embedded development. Due to the significant differences between these two IDEs in project creation, file format, compiler option, linked script, and debugging environment, special attention shall be paid to the compatibility in these areas in the migration process.

For the content of this chapter, please refer to the *AN1126_G32R501 Instructions for Use of G32R501 IDE and Tool Chain*.

10. Revision

Table 106 Document Revision History

Date	Version	Change History
January 2025	1.0	New
April 2025	1.1	Optimize descriptions

Statement

This document is formulated and published by Geehy Semiconductor Co., Ltd. (hereinafter referred to as “Geehy”). The contents in this document are protected by laws and regulations of trademark, copyright and software copyright. Geehy reserves the right to make corrections and modifications to this document at any time. Read this document carefully before using Geehy products. Once you use the Geehy product, it means that you (hereinafter referred to as the “users”) have known and accepted all the contents of this document. Users shall use the Geehy product in accordance with relevant laws and regulations and the requirements of this document.

1. Ownership

This document can only be used in connection with the corresponding chip products or software products provided by Geehy. Without the prior permission of Geehy, no unit or individual may copy, transcribe, modify, edit or disseminate all or part of the contents of this document for any reason or in any form.

The “极海” or “Geehy” words or graphics with “®” or “TM” in this document are trademarks of Geehy. Other product or service names displayed on Geehy products are the property of their respective owners.

2. No Intellectual Property License

Geehy owns all rights, ownership and intellectual property rights involved in this document.

Geehy shall not be deemed to grant the license or right of any intellectual property to users explicitly or implicitly due to the sale or distribution of Geehy products or this document.

If any third party’s products, services or intellectual property are involved in this document, it shall not be deemed that Geehy authorizes users to use the aforesaid third party’s products, services or intellectual property. Any information regarding the application of the product, Geehy hereby disclaims any and all warranties and liabilities of any kind, including without limitation warranties of non-infringement of intellectual property rights of any third party, unless otherwise agreed in sales order or sales contract.

3. Version Update

Users can obtain the latest document of the corresponding models when ordering Geehy products.

If the contents in this document are inconsistent with Geehy products, the agreement in the sales order or the sales contract shall prevail.

4. Information Reliability

The relevant data in this document are obtained from batch test by Geehy Laboratory or cooperative third-party testing organization. However, clerical errors in correction or errors caused by differences in testing environment may occur inevitably. Therefore, users should understand that Geehy does not bear any responsibility for such errors that may occur in this document. The relevant data in this document are only used to guide users as performance parameter reference and do not constitute Geehy's guarantee for any product performance.

Users shall select appropriate Geehy products according to their own needs, and effectively verify and test the applicability of Geehy products to confirm that Geehy products meet their own needs, corresponding standards, safety or other reliability requirements. If losses are caused to users due to user's failure to fully verify and test Geehy products, Geehy will not bear any responsibility.

5. Legality

USERS SHALL ABIDE BY ALL APPLICABLE LOCAL LAWS AND REGULATIONS WHEN USING THIS DOCUMENT AND THE MATCHING GEEHY PRODUCTS. USERS SHALL UNDERSTAND THAT THE PRODUCTS MAY BE RESTRICTED BY THE EXPORT, RE-EXPORT OR OTHER LAWS OF THE COUNTRIES OF THE PRODUCTS SUPPLIERS, GEEHY, GEEHY DISTRIBUTORS AND USERS. USERS (ON BEHALF OR ITSELF, SUBSIDIARIES AND AFFILIATED ENTERPRISES) SHALL AGREE AND PROMISE TO ABIDE BY ALL APPLICABLE LAWS AND REGULATIONS ON THE EXPORT AND RE-EXPORT OF GEEHY PRODUCTS AND/OR TECHNOLOGIES AND DIRECT PRODUCTS.

6. Disclaimer of Warranty

THIS DOCUMENT IS PROVIDED BY GEEHY "AS IS" AND THERE IS NO WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, TO THE EXTENT PERMITTED BY APPLICABLE LAW.

GEEHY'S PRODUCTS ARE NOT DESIGNED, AUTHORIZED, OR WARRANTED FOR USE AS CRITICAL COMPONENTS IN MILITARY, LIFE-SUPPORT, POLLUTION CONTROL, OR HAZARDOUS SUBSTANCES MANAGEMENT SYSTEMS, NOR WHERE FAILURE COULD RESULT IN INJURY, DEATH, PROPERTY OR ENVIRONMENTAL DAMAGE.

IF THE PRODUCT IS NOT LABELED AS "AUTOMOTIVE GRADE," IT SHOULD NOT BE CONSIDERED SUITABLE FOR AUTOMOTIVE APPLICATIONS. GEEHY ASSUMES NO LIABILITY FOR THE USE BEYOND ITS SPECIFICATIONS OR GUIDELINES.

THE USER SHOULD ENSURE THAT THE APPLICATION OF THE PRODUCTS COMPLIES WITH ALL RELEVANT STANDARDS, INCLUDING BUT NOT LIMITED TO SAFETY, INFORMATION SECURITY, AND ENVIRONMENTAL REQUIREMENTS. THE USER ASSUMES FULL RESPONSIBILITY FOR THE SELECTION AND USE OF GEEHY PRODUCTS. GEEHY WILL BEAR NO RESPONSIBILITY FOR ANY DISPUTES ARISING FROM THE SUBSEQUENT DESIGN OR USE BY USERS.

7. Limitation of Liability

IN NO EVENT, UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL GEEHY OR ANY OTHER PARTY WHO PROVIDES THE DOCUMENT AND PRODUCTS "AS IS", BE LIABLE FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, DIRECT, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE DOCUMENT AND PRODUCTS (INCLUDING BUT NOT LIMITED TO LOSSES OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY USERS OR THIRD PARTIES). THIS COVERS POTENTIAL DAMAGES TO PERSONAL SAFETY, PROPERTY, OR THE ENVIRONMENT, FOR WHICH GEEHY WILL NOT BE RESPONSIBLE.

8. Scope of Application

The information in this document replaces the information provided in all previous versions of the document.